# CONVEX PBUS I/O System
# Diagnostics Manual

Document No. 760-000750-203

Second Edition
October 1988

CONVEX Computer Corporation
Richardson, Texas USA

*CONVEX PBUS I/O System*
*Diagnostics Manual*
Order No. DHW-008
Second Edition



Printed in the United States of America

# Revision Sheet

*CONVEX PBUS I/O System*
*Diagnostics Manual*

| Edition | Document No. | Date | Description |
|---|---|---|---|
| Second | 760-000750-203 | Oct. 1988 | This release contains minor updates to all of the diagnostics. Several of the tape and disk diagnostics contain major revisions. |
| First | 760-000750-202 | June 1988 | First release. Contains all I/O and peripheral tests from the *C1 Diagnostics Reference Manual.* |

THIS PAGE INTENTIONALLY LEFT BLANK

# Table of Contents

## io4120 Multibus HSP/HIA Subsystem Test

## io5000 VMEbus I/O Processor Test

## dev4100 Multibus SMD Disk Test



## dev4110 Multibus SMD Disk Formatter, and Interactive Test

## dev4200 Multibus STC Tape Unit Controller Test

## dev4300 Multibus Terminal Controller Test

## dev4400 Multibus Line Printer Test



## dev4410 Multibus Plotter Test

## dev4500 Multibus Ethernet Controller Test



## dev4510 Multibus HYPERchannel Controller Test



## dev4600 Multibus Emulator Controller Test

## dev5130 VMEbus SMD/ESDI Disk Test and Formatter

dev5200 VME Tape Drive.

**dev5500 VMEbus Ethernet Controller Test**

# Appendixes



# List of Tables

# List of Figures

THIS PAGE INTENTIONALLY LEFT BLANK

# Preface

## Purpose and Intended Audience

This manual is divided into chapters that encompass the operation and interpretation of the various I/O system functional tests. This manual is not a tutorial, but rather a reference for the users of 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 diagnostics.

## Scope

This manual applies to all CONVEX computers.

## Outline

Each chapter in this manual covers a specific diagnostic function. To identify the general contents of each chapter, prefixes appear before the page number. The class and subtest descriptions and the invocation procedures of each test are covered within each chapter. The organization is as follows:

- **Diagnostics Environment** — Contains an introduction to 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
- **EGOS Overview** — Provides a brief overview of the Event Governed Operating System (EGOS) and how it relates to device and peripheral diagnostics testing
- **Dshell Overview** — Provides a brief overview of and a general introduction to the *dshell* utility
- **io** — Describes the IOP/VIOP and channel control unit tests
- **dev** — Describes all peripheral tests
- **Appendix A** — Contains information concerning how to use the *contact* facility to report problems

# Notational Conventions

The notational conventions used in this text are listed below:

- 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
- 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
- *Main memory* or *physical memory* is the physical storage installed in the processor
- *Logical memory* or *virtual memory* is the perceived amount of main 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
- TBD is an abbreviation for *To Be Determined*
- **Boldface** is user entered data

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. Warnings immediately precede the critical information and include a description of the hazard.

**CAUTION**

Cautions highlight procedures or information necessary to avoid damage to equipment, damage to software, or loss of data. Cautions immediately precede the critical information and include a description of the possible damage.

**NOTE**

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

# Associated Documents

Readers should become familiar with both the glossary of technical terms and the technical notation conventions listed in the preface. A feedback form is found in the rear of this handbook, and readers are invited to comment on the service and clarity of this text.

Other related topics are detailed in:

- *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-007
- *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 (C130, C210, C220)*, 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 obtain the most current version of any associated documentation, order using the product number. If the product number is not known, order by the exact title. In some situations the most current version is not desired. In order to receive a specific version of a manual, the manual must be ordered by a 12-digit document, or part, number, which can be provided by CONVEX.

The product number for this manual is DHW-008.
The document number for this manual is 760-000750-203.

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

# Hardware and Software Support

Hardware and software support can be obtained through the CONVEX Technical Assistance Center (TAC). The TAC can be reached in Texas by calling (214)952-0379, or by calling 1(800)952-0379 from other locations in the continental United States. Customers outside the United States should contact their local CONVEX office.

# 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.

### 1.1.1 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.

#### 1.1.1.1 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* | I/O subsystem related test |
| *mem* | Memory subsystem related test |
| *spu* | SP2 subsystem related test |

#### 1.1.1.2 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 |

#### 1.1.1.3 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 |

#### 1.1.1.4 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 |

#### 1.1.1.5 Current PBUS I/O System Test Program Name Assignments

See the following table for current PBUS I/O System test program name assignments:

**Table 1–5, Test Program Name Assignments**

| TEST PROGRAM NAME ASSIGNMENTS | |
|---|---|
| **Program Name** | **Test Name** |
| *dev4100* | Multibus SMD Disk Test |
| *dev4110* | Multibus SMD Disk Formatter, and Interactive Test |
| *dev4200* | Multibus STC Tape Unit Test |
| *dev4300* | Multibus Terminal Controller Test |
| *dev4400* | Multibus Line Printer Test |
| *dev4410* | Multibus Plotter Functional Test |
| *dev4500* | Multibus Ethernet Controller Test |
| *dev4510* | Multibus HYPERchannel Controller Test |
| *dev4600* | Multibus Emulator Controller Test |
| *dev5130* | VMEbus SMD/ESDI Disk Test and Formatter |
| *dev5500* | VMEbus Ethernet Controller Test |
| *io1000* | EPROM-Based IOP Self-Test |
| *io1200* | VIOP Self-Test |
| *io4000* | Multibus I/O Subsystem Test |
| *io4120* | Multibus HSP/HIA Subsystem Test |
| *io5000* | VMEbus I/O Processor Test |

# 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

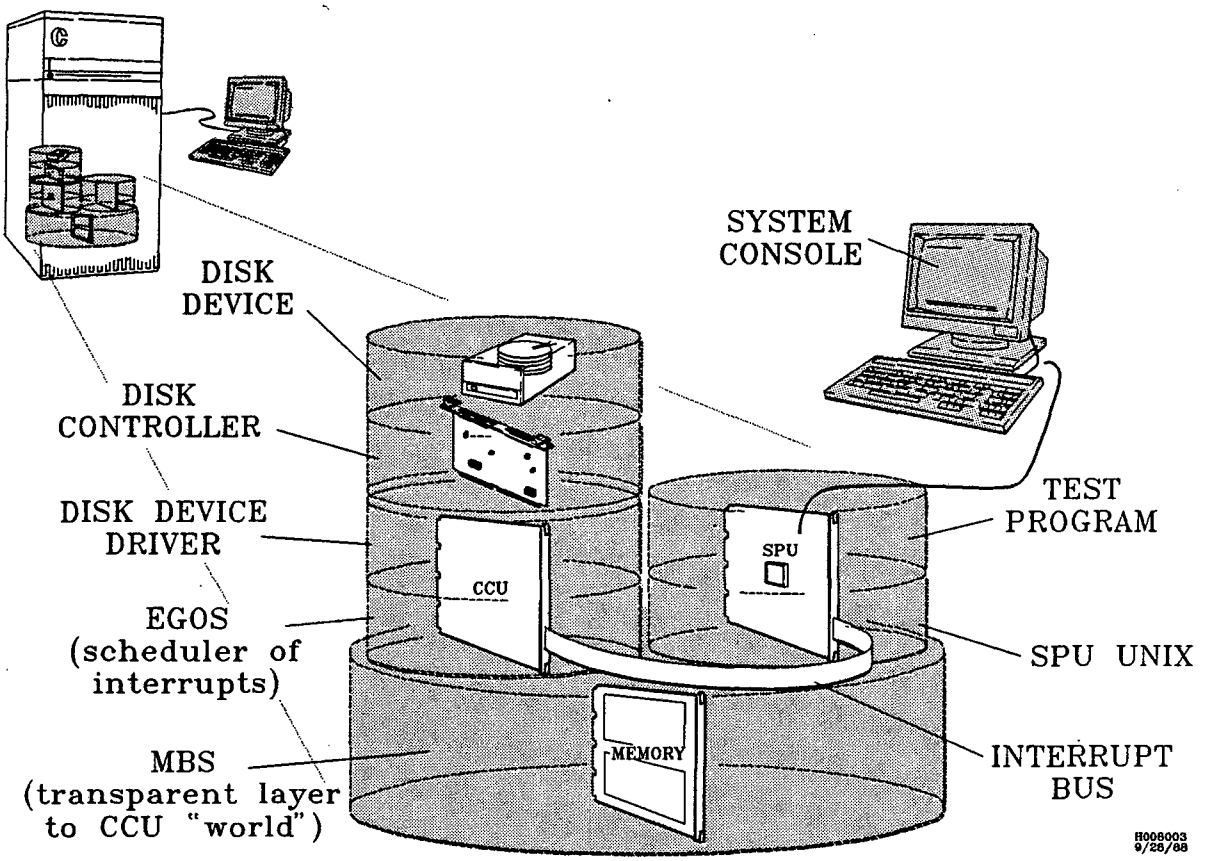
The VME interface version of EGOS is designed with a scheduler for the VIOP and is called VIOP EGOS. VIOP EGOS supports event driven device drivers as well as process type device drivers. VIOP EGOS utilizes a *sleep/wakeup* type of process control that improves efficiency of the device driver and makes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

# io4000

# Multibus I/O Subsystem Test

## Overview

The *io4000* test verifies the functionality of a specified IOP. The *io4000* test verifies the following on the IOP(s) specified:

- The 68000 on the IOP can correctly execute instructions.
- The IOP local memory is functional.
- The memory protection registers are operational.
- The cache memory can be written to and read from, and the associated cache control bits are functional.
- The four multibus interfaces operate in loopback mode.
- The IOP can boot a program from memory.
- The voltages on all installed Multibus chassis are within tolerance.

The functional test uses main memory as a communication port from the SPU to or from the IOP(s) being tested.

## Prerequisites and Required Equipment

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

**Table io4000–1, Hardware Requirements**

| C1, C120 | C200 Series |
|---|---|
| MCU | Memory System[1] |
| MAU | CPX |
| SPU | SP2 |
| IOP | IOP |
| MBCU | MBCU |
| | PIA |

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

# Test Invocation

The *io4000* 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 *io4000* 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 io4000–1, Test Invocation Sequence**

```
(spu)> cd /mnt/test RETURN
(spu)> sysreset RETURN
(spu)> mminit -s RETURN
(spu)> dshell RETURN
: test io4000 [-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 io4000** executes all *io4000* 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 powerup. 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 if 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 io4000–2, Alternate Test Invocation Sequence**

```
(spu)> cd /mnt/test RETURN
(spu)> initall RETURN
(spu)> dshell RETURN
: test io4000 [-c [class number(s)]] [-s [subtest number(s)]] [+> filename] RETURN
```

## 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. 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 io4000–3, 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

1: Enter IOP numbers [34567][1]                    (5) ->
2: Do you wish to run the multibus voltage tests?
   [y,n]                                           (n) ->
3: Enter OK, or :NN to return to question NN [OK]
                                                  (OK) ->
```

[1] The available selections for this prompt will vary depending on the machine architecture.

At any time during the test parameter sequence, several options are available as denoted at the top of the Test Parmeter 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

**?** — Request help for the current prompt (if available)

**^** — Return to the previous prompt

## Prompt Explanations

A description of the meaning of each prompt follows:

```
Enter IOP numbers [34567]                          (5) ->
```

This prompt allows for the selection of the IOP(s) to be scanned. (For more than one IOP, enter the numbers contiguously, for example 345.) After entering the IOP number(s), each IOP entered is scanned to see if it exists. Any invalid number causes the test to reprompt for valid IOP(s). The test executes only when all entries are valid.

```
Do you wish to run the multibus voltage tests?
[y,n]                                              (n) ->
```

This prompt selects whether Subtest 600, Multibus Voltage Test should be executed. If the default is selected, the multibus voltage test does not execute.

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

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

# Hardware Initialization Sequence

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

- Signals and interrupts for hard and soft errors are set
- Determine which Multibusses are present
- Determine whether the IOPs can run cache accelerated mode
- SPU local test variables are initialized
- The SPU verifies that the selected VIOPs are installed by scanning patterns into the VIOP LED register

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

# Class Descriptions

The *io4000* test contains the following four classes of subtests as shown in the following table. The individual class descriptions that follow list the time required to execute each subtest.

> **NOTE**
>
> The subtest execution times shown in this section were executed at 10Mhz.

**Table io4000–2, *io4000* Test Classes**

| CLASS | DESCRIPTION |
|---|---|
| 1 | 68000 subsystem tests |
| 2 | RAM pattern tests |
| 3 | Cache functionality tests |
| 6 | Multibus tests |

> **NOTE**
>
> The first time one of the subtests in the 300 - 600 range is executed, the IOP(s) under test must be loaded with the test program, which adds about 30 seconds to the test times.

## Class 1 Subtests

Class 1 subtests verify that the IOP can correctly load a program from main memory. These tests communicate with the SPU via the IOP LED register. The SPU places a command on the

LEDs through the IOP scan ring. The IOP then reads the LED register, executes the specified command, and places the result in the LED register.

In general, a return code of zero indicates the test completed without error. When the test fails, a nonzero value is returned, which identifies the step in the subtest that failed. The SPU reads this code and prints the appropriate diagnostic message. Class 1 subtests are listed in the following table:

**Table io4000–3, Class 1 Subtests**

| SUBTEST | DESCRIPTION | TIME (min:sec) |
|---|---|---|
| 100 | IOP Reset | :05 |
| 101 | IOP Self-test | :40 |
| 102 | IOP Initialization Command | :05 |
| 103 | IOP Boot Command | :05 |

**NOTE**

Subtest 100 must be run before subtests 101, 102, and 103. Also, subtest 103 cannot be looped.

**Subtest 100, IOP Reset**

Subtest 100 resets all the IOPs being tested.

**NOTE**

This subtest must be executed before subtests 101, 102, or 103.

Each IOP then performs the following:

- Tests operation of the instructions:
  - beq
  - bne
  - move <ea>,cc
  - cmpi #<data>,<ea> for all operand lengths
  - cmpa <ea>,an
  - cmp<ea>,dn
- Tests uniqueness of registers a0, d0-d1
- Tests functionality of all bits in a0, d0-d1
- Tests ability to calculate a checksum

If all of the previously listed tests pass, then the IOP performs a checksum test on the EPROM. If the checksum test passes, the IOP then enters a loop waiting for commands from the SPU to be passed through the IOP scan ring. If any of the above tests fail, an error signal is sent back to

the SPU via the send-error bit in the miscellaneous diagnostic register, and the IOP 68000 performs a double-bus fault and dies.

### Subtest 101, IOP Self-test

Subtest 101 places a self-test command on the red LEDs of the IOPs under test. Each IOP being tested then performs a series of seven tests (CPU2, RAM1, CPU3, RAM2, MAPTST, CACHETST, and MBLBTST). The following are descriptions of the tests:

#### CPU2

This test checks register functionality for the various 68000 registers, checks the addressing modes, verifies the jump and branch instructions, performs integer arithmetic tests, data movement operations, shift and rotate instructions, and bit-manipulation operations.

#### RAM1

This test determines the size of local memory by writing to the first word in each bank until a bus error occurs. It then performs a walking '1's and '0's test on the first word after the ROM and on the first word of all the following banks. Then, on the last 4 Kbytes of installed local memory, it performs a uniqueness test and a true/complement pattern test. Memory parity is checked by writing a word with inverted parity and then reading it to generate a parity error. The word is then rewritten with the correct parity.

#### CPU3

This test checks the 68000 instructions not tested in subtest 100 or CPU2 by using the last 4 Kbytes of installed local memory as a stack. It also checks privilege violations and address errors. The *stop*, *trace*, and *reset* instructions are not checked.

#### RAM2

This test is similar to RAM1 except that it checks local memory from immediately after the ROM to the word preceding the last 4 Kbytes of installed local memory. A parity test is not performed.

#### MAPTST

This test performs a walking '1's and '0's test on the memory protection register at `0xff8000`. Then, it performs a uniqueness test and a true/complement pattern test for all the 256 registers. Next, it copies ROM to local memory and modifies the protection for installed local memory to valid, read, write, and execute. It clears the registers for uninstalled local memory. Then, memory protection is turned on, and the valid, read, write, and execute bits are verified from supervisor and user state. Finally, at the end of the test, memory protection is again disabled.

#### CACHETST

This test clears the cache after disabling cache parity checking and clearing the longword or byte dirty flags and the tag flag registers. Then, it reclears the tag flag registers and enables cache parity checking. Any pending cache or PBUS interrupts are cleared, and

cache interrupts are enabled. The local memory copy of the EPROM is mapped in, and a checksum is performed on it. Next, the test sets the map registers to point to the last Mbyte of the PBUS. A unique pattern is written to the first 64 shortwords in each main memory 4 Kbytes window. As each word is written, the dirty bits are checked to see that they are updated appropriately. The test then clears the longword or byte dirty flags and marks all tag registers as loaded. Finally, the pattern written to each cache location is verified.

**MBLBTST**

This test resets the Multibus cable drivers and places them in loopback mode. In each Multibus window, the local memory copy of the EPROM is mapped in, and a checksum test is performed. The pattern left by the cache test is then checked through each of the Multibus windows.

### Subtest 102, IOP Initialization Command

Subtest 102 determines the size of the installed local memory, and then clears local memory from the word following the EPROM to the end. The test then disables cache parity checking, and for each of the map registers performs the following:

- Clears the dirty 'a' and 'b' bits
- Clears the corresponding 128 cache bytes
- Clears the dirty 'a' and 'b' bits again

Next, the test enables cache parity checking, and points the first map register to the population configuration map on the MCU. The Population Configuration Map (PCM) is searched in order to locate the first 2 Mbytes of main memory. Once located, the map registers are initialized to point to the first half of this 2 Mbyte block. The motherboard slot number is used to index into a table in main memory. This table contains a pattern, previously initialized by the SPU, that is to be echoed by the IOP. The pattern is the current time, in seconds, logically 'anded' with a mask of `0xffffff00`. When the IOP echoes this pattern, the subtest ends.

### Subtest 103, IOP Boot Command

> **NOTE**
>
> Before executing Subtest 103, Subtest 102 must have completed successfully. This subtest checks only the SPU *load* command. Also, it is not possible to loop on this subtest.

Initially, Subtest 103 determines the size of local memory, and copies the EPROM to the last 16 Kbytes of installed local memory. A checksum is performed on local memory to verify it. The test sets memory protection for installed local memory to valid, read, write, and execute. The memory protection registers for uninstalled local memory are cleared. The interrupt status register is then cleared, and the IOP jumps to the local memory copy of the EPROM. Memory protection is enabled. The IOP slot number is used as an index into a table in main memory. The table is located in the first Mbyte of installed main memory.

After the test begins, the IOP enters a loop while waiting for the SPU to place commands in main memory. The valid commands are listed in the following table:

**Table io4000–4, Valid SPU Commands for Subtest 103**

| COMMAND | DESCRIPTION |
|---|---|
| *load* | Copies main memory to IOP memory, a byte at a time. As each byte is moved, *load* tallies it into a checksum. After main memory is moved, it checks for parity errors. If no errors, the checksum is placed in main memory; otherwise, negation of calculated checksum goes in main memory. At completion, *load* waits for more commands. |
| *reset* | Simulates a reset by loading the supervisor-stack pointer from memory location 0 and the *pc* from memory location 4. |
| *jump* | Jumps to the address specified in main memory. The *ssp* is set to the top of installed local memory. |

## Class 2 Subtests

Class 2 subtests are listed in the following table:

**Table io4000–5, Class 2 Subtests**

| SUBTEST | DESCRIPTION | TIME (min:sec) |
|---|---|---|
| 200 | PBUS Interrupt | :02 |
| 201 | PBUS Test-and-set | :02 |
| 202 | IOP Memory Access | :05 |

### Subtest 200, PBUS Interrupt

Subtest 200 verifies the ability of the IOP to request and use the PBUS interrupt bus. The subtest consists of the following three steps:

**Step 1:** Interrupt Receive Test

In the receive test, all 256 PBUS interrupts are sent by the SPU to the IOP, one at a time. After each interrupt is sent, the IOP verifies that only one interrupt was received and that the interrupt was received by the proper group. The group is determined by taking the modulo 4 residue of the interrupt number.

**Step 2:** Interrupt Transmit Test

In the interrupt transmit test, the IOP sends the SPU interrupt numbers 8, 9, 10, and 11 (the four interrupts that the SPU is capable of receiving). The IOP verifies that the acknowledge for each interrupt is received and the SPU verifies the reception of the four interrupts.

**Step 3:** Interrupt Loopback Test

The interrupt loopback test causes all 256 interrupts to be sent and received by the IOP in each of the four possible groups. Like the receive test, as each interrupt is sent, the IOP verifies that only one interrupt is received and that the acknowledge is also received.

> **NOTE**
>
> The terms receive and transmit are from the IOP's aspect.

### Subtest 201, PBUS Test-and-set

Subtest 201 verifies that all PMAP registers operate in test-and-set mode by placing each PMAP register in test-and-set mode in sequence. The PMAP register that logically follows the register under test is prepared for non test-and-set operation and it verifies the operation.

A byte in the main memory command table is set to zero through the check window. It is read through the test window and the value returned is checked for a zero. The check window reads the byte in main memory and the value read is expected to be `0xff`. A second access through the test window verifies that the returned value is also `0xff`.

### Subtest 202, IOP Memory Access

Subtest 202 verifies the IOP's ability to access all of allocated main memory. The test determines which blocks in memory are allocated by reading the PCM on the SPU, memory is allocated in 2 Mbyte blocks. The test writes and then reads a single unique integer value in each allocated block, and the test then re-reads each of the blocks to verify proper addressing.

## Class 3 Subtests

Class 3 subtests verify the functionality of the IOP cache memory. The current subtests verify the cache in accelerated mode, normal mode, and in bypass mode.

> **NOTE**
>
> Not all of the subtests in Class 3 are currently implemented. Only the subtests that are currently available are listed in the following table.

**Table io4000–6, Class 3 Subtests**

| SUBTEST | DESCRIPTION | TIME (min:sec) |
|---|---|---|
| 300 | IOP Cache Accelerate Read | 3:00 |
| 301 | IOP Cache Accelerate Write | 5:00 |
| 302 | IOP Cache Bypass Read | 4:00 |
| 303 | IOP Cache Bypass Write | 4:00 |

### Subtest 300, IOP Cache Accelerate Read

First subtest 300 sets the accelerate bit in the PMAP register. Then it sets the PMAP register that logically follows the window under test to nonaccelerate, nonbypass mode.

The IOP slot number is used to index into a table in main memory that contains patterns and pointers to main memory, which are used to test the accelerate function. The SPU initializes the table before the accelerate test command is given to the IOP(s). The table is accessed through the nonaccelerated PMAP register as described in the previous paragraph. The table has checksums and an IOP ID that are checked to ensure the integrity of the data.

Using the information from the table, the test checks the PMAP window under test. Before any location in the page under test is referenced, an initial 64-byte pattern corresponding to buffer locations `0` to `3f` is initialized to a known pattern through the nonaccelerated window. The pattern used for the first byte in the page is the IOP slot number plus the page number of the page under test, modulo 256. The test initializes each successive byte to the value of the previous byte plus 1, modulo 256.

Then a loop is entered in which each of the 4096 bytes in the page is read and checked for valid data. Additionally, each time byte `0` of each of the two cache buffers is read, the PTAG register for the page is checked to ensure the following:

- The accelerate-on reference bit is set.
- The loaded bit is set.
- The buffer dirty bit is clear.
- The TAG value and its parity are properly set.

Then, if the previous conditions are met, the test uses the nonaccelerated window to negate the current pattern in main memory. The original pattern remains in the cache. The pattern for the next 64-byte group then is initialized through the nonaccelerated window. When byte `1` of a buffer is read, the test checks the following:

- The accelerate-on reference bit is clear.
- The loaded bit is set for both buffers.
- The TAG value for the current buffer has not changed, and the TAG value for the other buffer is properly set.

### Subtest 301, IOP Cache Accelerate Write

Subtest 301 initializes the main memory area used for the write test to the negation of the pattern to be written there. The test then uses the sum of the IOP slot number plus the page number of the page under test, modulo 256, as the pattern for the first byte. Next, it enters a loop into which each of the 4096 bytes in the page is written, one byte at a time. As each byte is written, the dirty bits for that longword are checked to make sure that they are set to the expected value. The test then checks the PTAG register for valid data. When byte `0x3f` of each buffer is written, the corresponding area of main memory is read through the nonaccelerated window to ensure that the pattern has *not* yet been written.

Every time byte 0 of a cache buffer is written, beginning with byte `0x40` in the page under test, the PTAG register is checked to verify that the accelerate-on reference bit is set. When byte 1 of a buffer is written, the nonaccelerated window reads main memory to verify that the pattern for the previous 64-byte block is written to memory and to verify that the accelerate-on reference bit resets.

After the last 64-byte block is written, the window under test is flushed so that all data is written to memory. Finally, the test again checks the pattern in the entire 4096-byte block for integrity through the nonaccelerated window.

### Subtest 302, IOP Cache Bypass Read

Subtest 302 tests each of the windows in the IOP cache. As in the accelerate read test, this test places the window that logically follows the window under test in nonaccelerate, nonbypass mode and verifies the operation of the window under test. Then it initializes the main memory area for the bypass test to the negation of the pattern to be written there. It uses the same patterns as those listed for Subtest 300, IOP Cache Accelerate Read Test.

For each of the 4096 bytes in the window, the test first uses the nonaccelerate window to write a one byte pattern to main memory. Then it reads the byte just written through the window under test and verifies the data. Finally it negates the byte in main memory through the nonbypass window.

### Subtest 303, IOP Cache Bypass Write

Subtest 303 , the logical complement of Subtest 302, tests each window in the IOP cache. It initializes the main memory area used to test the cache to the negation of the pattern that is to be written there. It uses the same pattern described in Subtest 301, IOP Cache Accelerate Write Test.

For each of the 4096 bytes in the window, the test first uses the window under test to write a 1-byte pattern to main memory. Then it uses the nonbypassed window to verify that the pattern was written to main memory.

## Class 6 Subtests

Class 6 subtests test the operation of portions of the Multibus Control Units (MBCUs) that are installed on the IOP(s) being tested. The system I/O configuration file, */ioconfig*, is read to determine what MBCUs are present.

**NOTE**

Not all of the subtests in this class are currently implemented. The subtest that is currently available is listed in the following table.

**Table io4000–7, Class 6 Subtest**

| SUBTEST | DESCRIPTION | TIME (min:sec) |
|---|---|---|
| 600 | Multibus Voltages | :05 |

**Subtest 600, Multibus Voltages**

Subtest 600 reads and tests the four voltages of each Multibus that is present for out-of-tolerance conditions. The voltages and tolerances for a Multibus are listed in the following table:

**Table io4000–8, Multibus Voltages and Tolerances**

| VOLTAGE | MINIMUM TOLERANCE | MAXIMUM TOLERANCE |
|---|---|---|
| +12.00 | +11.40 | +12.60 |
| +05.00 | +04.75 | +05.25 |
| -05.00 | -05.75 | -04.75 |
| -12.00 | -12.60 | -11.40 |

THIS PAGE INTENTIONALLY LEFT BLANK

# io4120

# Multibus HSP/HIA Subsystem Test

## Overview

The *io4120* test verifies the operation of a High Speed Parallel interface (HSP) and HSP Interface Adapter (HIA).

The *io4120* test verifies the following local to the HSP:

- 68000 instruction execution
- HSP local memory operations
- HSP local memory protection
- Input bus integrity
- PBUS header generation
- PBUS access verification
- PBUS interrupt operations
- Correct HSP ATU operations in both physical and virtual address modes
- ATU error detection
- Output bus integrity
- Error status register verification

Additionally, the *io4120* test checks these functions that require the HIA:

- Register accesses
- Interface interrupt test
- Interface parity generation or detection
- Synchronous data transfers in both physical and logical address modes
- Bus arbitration
- Clock selection
- Channel interleave
- Error detection and recovery

## Prerequisites and Required Equipment

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

**Table io4120–1, Hardware Requirements**

| C1, C120 | C200 Series |
|---|---|
| MCU | Memory System[1] |
| MAU | CPX |
| SPU | SP2 |
| HIA | HIA |
| HSP | HSP |
| FSE | FSE |
| | PIA |

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

# Test Invocation

The *io4120* 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 *io4120* 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 io4120–1, Test Invocation Sequence**

```
(spu)> cd /mnt/test RETURN
(spu)> sysreset RETURN
(spu)> mminit -s RETURN
(spu)> dshell RETURN
: test io4120 [-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 io4120** executes all *io4120* 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 powerup. 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 if 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 io4120–2, Alternate Test Invocation Sequence**

```
(spu)> cd /mnt/test RETURN
(spu)> initall RETURN
(spu)> dshell RETURN
: test io4000 [-c [class number(s)]] [-s [subtest number(s)]] [+> filename] RETURN
```

## 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.

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 io4120–3, 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

Hsp ccu slot number [0-3]                          (2) ->
Hsp default clock rate [0-3]                       (0) ->
Hia outclk default rate [0-3]                      (0) ->
Hia busclk default rate [0-1]                      (0) ->
Channel mask for compliance test - ls bit is chnl 0
[0x0-0xffff]                                  (0xffff) ->
Default power margin for HIA chassis (+5 only)
[l,n,u]                                            (n) ->
ATU scheme to test ('1' for C1, '2' for C2, 'b' for both)
[1,2,b]                                            (b) ->
Enter OK, or :NN to return to question NN [OK]
                                                  (OK) ->
```

At any time during the test parameter sequence,.several options are available as denoted at the top of the Test Parmeter 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

**?** — Request help for the current prompt (if available)

**^** — Return to the previous prompt

## Prompt Explanations

Each prompt is repeated and explained in the following paragraphs.

> **NOTE**
>
> The CCU slot specified is checked via the diagnostic bus to ensure that the HSP is present. Any invalid entry reprompts for the HSP slot number.

```
Hsp ccu slot number [0-3]                          (2) ->
```

Enter the CCU slot number the HSP resides in.

```
Hsp default clock rate [0-3]                    (0) ->
```

Enter the HSP default clock rate which sets the transfer rate from the HSP to the HIA. Enter **0** to select the highest rate.

```
Hia outclk default rate [0-3]                   (0) ->
```

Enter the HIA outclk default rate which sets the transfer rate from the HIA to the HSP. Enter **0** to select the highest rate.

```
Hia busclk default rate [0-1]                   (0) ->
```

Enter **0** or (RETURN) to select the default 40 megahertz crystal. This crystal produces a 10 megahertz transfer rate. Enter **1** to select a user defined crystal. A user defined crystal, for example a 36 megahertz crystal producing a 9 megahertz transfer rate, would be selected.

```
Channel mask for compliance test - ls bit is chnl 0
[0x0-0xffff]                                (0xffff) ->
```

Enter **0xffff** to select an HIA. Refer to Subtest 4999, CONVEX Register Compliance Test for more information.

```
Default power margin for HIA chassis (+5 only)
[l,n,u]                                         (n) ->
```

Enter the power margin for the HIA chassis. The power margin may be changed to help debug hardware problems by forcing intermittent errors to hard errors.

- Enter **l** to select the lower power margin, 4.75 volts.
- Enter **n** to select the nominal power margin, 5.00 volts.
- Enter **u** to select the upper power margin, 5.25 volts.

```
ATU scheme to test ('1' for C1, '2' for C2, 'b' for both)
[1,2,b]                                         (b) ->
```

Enter **1** if the test is to execute on a C1, C120 machine. Enter **2** if the test is to execute on a C200 Series machine. Enter **b** for both ATU schemes to test on a machine.

> **NOTE**
>
> If **b** is entered, some subtests will take twice as long to execute.

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

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

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 io4120–4, Sample Test Parameter Summary**

```
                TEST PARAMETER SUMMARY

Hsp ccu slot number                          : 2
Hsp default clock rate                       : 0
Hia outclk default rate                      : 0
Hia busclk default rate                      : 0
Channel mask for compliance test - ls bit is chnl 0
                                             : 0xffff
Default power margin for HIA chassis (+5 only)
                                             : n
ATU scheme to test ('1' for C1, '2' for C2, 'b' for both)
                                             : b
Enter OK, or :NN to return to question NN : OK
```

# Hardware Initialization Sequence

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

- Calls routine to read PCM to determine where main memory is
- Allocates SPU windows for use in test
- Resets system through Scan Interface
- Turns on clocks for HSP
- Sets up software to be aware of hard and soft errors
- SPU local test variables are initialized
- Displays a Test Parameter Summary

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

# Class Descriptions

The *io4120* test has four classes of subtests as shown in the following table:

**Table io4120–2, *io4120* Test Classes**

| CLASS | DESCRIPTION |
|---|---|
| 1 | 68000 subsystem tests |
| 2 | HSP stand alone tests |
| 4 | HSP/HIA tests |
| 5 | HSP/HIA/FSE tests |

## Class 1 Subtests

Class 1 subtests verify that the HSP can correctly load a program from main memory. Class 1 subtests reside in the HSP's EPROM and communicate with the SPU via the LED and the auxiliary test result registers on the HSP scan ring.

> **NOTE**
>
> The following are restrictions on the execution sequence of these subtests:
>
> - Subtest 100 must be the first subtest in this class executed
> - Subtests 101 and 102 may be executed in any order and may be executed in a loop until a failure occurs. If a failure occurs, Subtest 100 must be reexecuted.
> - After Subtest 103 is executed, Subtest 100 must be reexecuted.
>
> It is recommended that these tests be executed in numerical order.

The Class 1 subtests are listed in the following table:

**Table io4120–3, Class 1 Subtests**

| SUBTEST | DESCRIPTION | TIME (min:sec) |
|---|---|---|
| 100 | HSP Reset | <:02 |
| 101 | HSP Self-test | 01:20 |
| 102 | HSP Memory Initialization | <:02 |
| 103 | HSP Boot | <:02 |

### Subtest 100, HSP Reset

Subtest 100 resets the HSP via the scan ring. This causes the HSP to execute the initialization code in its EPROM. The initialization code saves the state of the HSP as found on reset and verifies that the HSP's 68000 can calculate a checksum. When verified, the HSP calculates a checksum of the EPROM. If these tests pass, the HSP enters a loop and waits for an EPROM command to be placed on the LED register. If this test fails, the HSP informs the SPU by asserting a hard error (a double bus fault) which causes the 68000 to stop executing instructions.

### Subtest 101, HSP Self-test

Subtest 101 is the HSP's self-test and consists of 10 steps that verify the HSP's ability to load a program from main memory. The SPU commands the HSP to perform this test by placing a code in the HSP's LED registers via the scan ring. When this code is detected, the HSP performs the steps listed below. If a failure is detected, the HSP returns the step number and other

relevant data via the LED and auxiliary test result register.

**Step 1:** Verify Instructions Needed for Memory Test

Step 1 performs the following:

- Test HSP 68000 registers
- Check addressing modes
- Verify jump and branch instructions
- The following operations are executed:
  - Integer arithmetic tests
  - Data movement operations
  - Shift and rotate instructions
  - Bit manipulation

**Step 2:** Initial Memory Operations Test

Step 2 determines the size of memory by writing to the first word in each bank until a bus error occurs. A walking '1' and '0' test is performed on the first word after the ROM, and on the first word of the following banks. Then a uniqueness test and a true/complement pattern test is performed on the last 4 Kbytes of installed memory. Memory parity is checked by writing a word with inverted parity and then reading it to generate a parity error. The word is then rewritten with the correct parity.

**Step 3:** Verify Remaining 68000 Instructions

Step 3 uses the last 4 Kbytes of installed memory as a stack and checks the 68000 instructions not tested in Subtest 100 or in step 1 of this subtest. However, the *stop*, *trace*, and *reset* instructions are not checked. Step 3 also checks privilege violations and address errors.

**Step 4:** Test Remaining Memory

Step 4 is similar to step 2, but it checks memory from immediately after the ROM, to the word preceding the last 4 Kbytes of installed memory. A parity test is not performed.

**Step 5:** Verify HSP Local Memory Protection

Step 5 performs a walking '1' and '0' test on the memory protection register at `0xff8000`. A uniqueness test and a true/complement pattern test is performed on all the 256 registers. The test copies the HSP's ROM into its memory, modifies the protection for installed memory to valid, read, write, and execute. Then the registers are cleared for uninstalled memory. Memory protection is turned on, and the valid, read, write, and execute bits are verified from the supervisor state and from the user state. At the end of the test, memory protection is disabled.

**Step 6:** Input Bus Test

Step 6 tests input bus integrity by pattern testing a longword in ATU memory. ATU memory is in two different address spaces in the HSP. The first address space checks the lower 32 bits of the input bus, and the second checks the upper 32 bits. The same 32 bit

memory location is checked in both address spaces.

**Step 7:** PMAP Register Test

Step 7 tests the memory that stores the Pmap register data by performing column functionality, uniqueness, and bit functionality tests. Bad parity detection is then checked. Then a functionality test of the valid bit in the Pmap register is performed.

**Step 8:** Not Used

Step 8 is not used in the self-test in order to be able to distinguish a failure of a given self-test step from a HSP that has not started execution of its self-test. The value of 8 is used by the SPU to command the HSP to perform its self-test. As the HSP's self-test executes, the value is incremented. By skipping step 8, the SPU can determine if the HSP started execution of its self-test.

**Step 9:** PBUS Header Generation

Step 9 checks the HSP's PBUS header generation function by asserting the PBUS_TEST bit in the FDR register. Headers are generated for physical addresses of zero, all ones, walking zero, and walking one for both reads and writes in 8 and 16 bit access modes. When a header is generated, it is verified by inspection of the PBUS_HDR register. PBUS data is not transferred in during this test.

**Step 10:** PBUS Access Test

Step 10 performs a PBUS access test which reads, verifies, and echos a pattern placed in main memory by the SPU. This test verifies that the SPU and the HSP agree on the location of main memory and on the proper byte ordering in that memory. On C100 machines, the first access to the PBUS made by the HSP is to read the PCM. The PCM is used to locate the test pattern in main memory. On C200 machines, a register on the PIA is read to locate the test pattern in main memory.

### Subtest 102, HSP Memory Initialization

Subtest 102 initializes the HSP's memory after the local memory size is determined. The local memory is cleared from the word following the end of the EPROM to the end of installed memory. ATU memory and the device buffer memory is then initialized. The first PMAP register is pointed to the PCM and it is searched for the first 2 Mbytes of physical memory. Once located, the map registers are initialized to point to the first Mbyte of installed memory. The CCU slot number is used to index into a table in main memory. This table contains a 32 bit pattern initialized by the SPU. The HSP reads this pattern and returns it to main memory. The returned pattern is verified by the SPU. The pattern is the current time, in seconds, logically 'anded' with a mask of `0xffffff00` and logically 'ored' with the CCU slot number.

**Subtest 103, HSP Boot**

> **NOTE**
>
> Before executing Subtest 103, Subtest 102 must have completed successfully. It is not possible to loop on this subtest.

Subtest 103 determines the size of local memory, and copies the EPROM to the last 16 Kbytes of installed local memory. A checksum is performed on local memory to verify it. The test sets memory protection for installed local memory to valid, read, write, and execute. The memory protection registers for uninstalled local memory are cleared. The interrupt status register is then cleared, and the HSP jumps to the local memory copy of the EPROM. Memory protection is enabled. The HSP slot number is used as an index into a table in main memory. The table is located in the first Mbyte of installed main memory.

After the test begins, the HSP enters a loop while waiting for the SPU to place commands in main memory. The valid commands are listed in the following table:

**Table io4120–4, Valid SPU Commands for Subtest 103**

| COMMAND | DESCRIPTION |
|---|---|
| *load* | Copies main memory to HSP memory, a byte at a time. As each byte is moved, *load* tallies it into a checksum. After main memory is moved, it checks for parity errors. If no errors, the checksum is placed in main memory; otherwise, negation of calculated checksum goes in main memory. At completion, *load* waits for more commands. |
| *reset* | Simulates a reset by loading the supervisor-stack pointer from memory location 0 and the *pc* from memory location 4. |
| *jump* | Jumps to the address specified in main memory. The *ssp* is set to the top of installed local memory. |

> **NOTE**
>
> Subtest 103 checks only the *load* command.

## Class 2 Subtests

Class 2 subtests test the HSP without a device attached to the HSP. This class of subtests is loaded into the HSP's local RAM from main memory. The following functions are tested:

- PBUS interrupt operations
- The HSP's address translation

- Address translation error detection
- Output bus integrity
- Error status register verification
- Line clock interrupt operation

The following table lists the Class 2 subtests:

**Table io4120–5, Class 2 Subtests**

| SUBTEST | DESCRIPTION | TIME (min:sec) |
|---|---|---|
| 210 | PBUS Interrupt | 00:36 |
| 220 | PBUS Test-and-set | <:02 |
| 230 | ATU Memory Pattern | <:02 |
| 231 | CSR Memory Pattern | <:02 |
| 232 | Output Bus | <:02 |
| 233 | Device Buffer Memory Pattern | <:02 |
| 240 | ATU Local Translation - physical addresses | <:02 |
| 241 | ATU Local Translation - virtual addresses | <:02 |
| 250 | ATU Virtual Address Translation | <:02 |
| 251 | ATU Memory Parity Error Detection | <:02 |
| 252 | ATU *pte* Error Detection | <:02 |
| 270 | Status Register Verification | <:02 |
| 280 | Line Clock Interrupt | 00:10 |

**Subtest 210, PBUS Interrupt**

Subtest 210 verifies the HSP's ability to request and use the PBUS's interrupt bus. It consists of three steps, an interrupt receive test, an interrupt transmit test, and an interrupt loopback test.

> **NOTE**
>
> The terms receive and transmit are from the HSP's aspect.

**Step 1:** Receive Test

In the receive test, all 256 PBUS interrupts are sent by the SPU to the HSP, one at a time. After each interrupt is sent, the HSP verifies that only one interrupt was received and that the interrupt was received by the proper group. The group in which each interrupt is received is determined by taking the modulo 4 residue of the interrupt number.

**Step 2:** Interrupt Transmit Test

In the interrupt transmit test, the HSP sends the SPU interrupt numbers 8, 9, 10, and 11, the four interrupts that the SPU is capable of receiving. The HSP verifies that the acknowledge for each interrupt is received and the SPU verifies the reception of the four interrupts.

**Step 3:** Interrupt Loopback Test

The interrupt loopback test causes all 256 interrupts to be sent and received by the HSP in each of the four possible groups. Like the receive test, as each interrupt is sent the HSP verifies that only one interrupt is received and that the acknowledge is also received.

### Subtest 220, PBUS Test-and-set

Subtest 220 verifies that all Pmap registers operate in test-and-set mode by placing each Pmap register in test-and-set mode in sequence. The Pmap register that logically follows the register under test is prepared for nontest-and-set operation and it verifies the operation.

A byte in the main memory command table is set to zero through the check window. It is read through the test window and the value returned is checked for a zero. The check window reads the byte in main memory and the value read is expected to be `0xff`. A second access through the test window verifies that the returned value is also `0xff`.

### Subtest 230, ATU Memory Pattern

Subtest 230 performs column functionality, uniqueness, and bit functionality tests on the memory used by the ATU controller. It verifies that memory parity checking is operational by writing a byte with bad parity, reading it, and verifying that a parity interrupt occurs.

Since the Pmap registers and the ATU share a common memory, a uniqueness test over the two address spaces is also performed.

### Subtest 231, CSR Memory Pattern

Subtest 231 tests the channel status memory by performing column functionality, uniqueness, and bit functionality tests. There is no parity on the CSR memory.

### Subtest 232, Output Bus

Subtest 232 checks the output bus by pattern testing two 32 bit words in the buffer memory used in device transfers. The first 32 bit location tests the bus in bit locations 0 to 31, and the second checks it in bits 32 to 63. Patterns of zero, all ones, alternating ones and zeros, walking ones, and walking zero are used in the subtest.

### Subtest 233, Device Buffer Memory Pattern

Subtest 233 checks the device buffer memory for column functionality, uniqueness, and bit functionality. Parity is checked in the same way as in Subtest 230, ATU Memory Pattern.

### Subtest 240, ATU Physical Address Mode Header

Subtest 240 checks the ATU controller's ability to format PBUS headers while in physical address mode. The 68000 uses the FDR register's control bits to instruct the ATU to create PBUS

headers.

ATU memory is initialized with the start address and byte count field such that the PBUS header is loaded with all zeros, all ones, a walking one and a walking zero. As each header is generated, the ATU_OREG register is inspected for the correct header.

This subtest asserts the PBUS_TEST bit in the FDR register to inhibit the generated headers from being sent to the MCU when the ATU is not functioning correctly.

### Subtest 241, ATU Virtual Address Mode Header

Subtest 241 uses the FDR register to force the ATU to construct PBUS headers. The ATU's memory is initialized so that the ATU constructs headers for data transfers, *pte* transfers and *sdr* transfers in increasing levels of complexity. This is done by walking a one in the logical address that the ATU is translating. As each address is translated, the ATU_OREG register or the PBUS_HDR register as appropriate is checked for valid header contents.

This subtest asserts the PBUS_TEST bit in the FDR register to inhibit the generated headers from being sent to the MCU.

### Subtest 250, ATU Virtual Address Translation

Subtest 250 checks the ATU's logical address translation. The SPU fills a set of *sdr*s and *pte*s in main memory with pointers to nonexistent main memory. As the HSP walks a one through the logical address, it initializes only those locations in main memory and ATU memory that are required to complete the translation. As each translation occurs, the HSP verifies that the ATU only modified the needed locations in its memory. The *sdr*s and *pte*s in main memory are then restored to their initial state before the next logical address is checked.

Subtest 250 is executed for each ATU channel.

### Subtest 251, ATU Memory Parity Error Detection

Subtest 251 tests the ATU's parity error detection. It is tested by introducing a memory parity error into ATU register 6, the next address register, starting a translation, and verifying the error was logged in the CSR after the translation.

### Subtest 252, ATU *pte* Error Detection

All *pte* entries used for HSP logical address translation must have bit 11 set to show that the selected *pte* is valid for I/O. Subtest 252 verifies that this bit is being checked by placing a *pte* level 2 in main memory without the bit set. A translation is started that reads the corrupted *pte*. The CSR register is checked for the global channel error flag set, as well as the I/O protection error.

The subtest is repeated with a corrupted *pte* level 1.

### Subtest 270, Status Register Verification

Subtest 270 checks HSP error log recording functions by forcing various error conditions and verifying that they have been correctly reported. The subtest forces read, write, and execute errors on HSP local memory and verifies that they are reflected in the BERRLOG and ADDRHI and ADDRLO registers.

PBUS read, write, and test and set errors are generated and the BERRLOG, PBERRLOG, and ADRHI and ADRLO registers are verified.

Finally, a DTACK timeout error is forced and the BERRLOG and ADRHI and ADRLO registers are verified.

### Subtest 280, Line Clock Interrupt

Subtest 280 enables the line clock interrupt on the HSP and waits for 600 line clock interrupts to occur. The SPU UNIX time of day is checked to ensure that the 600 interrupts took 10 seconds to occur.

## Class 4 Subtests

Class 4 subtests verify the functionality of the HSP/HIA interface. These subtests are loaded into the HSP's local memory from main memory and include tests for the following:

- Asynchronous and synchronous data transfers
- Address translation
- Clock selection
- Error detection and recovery

**Table io4120–6, Class 4 Subtests**

| SUBTEST | DESCRIPTION | TIME (min:sec) |
|---|---|---|
| 4100 | HIA Reset | 00:22 |
| 4101 | HIA Voltage | 00:22 |
| 4110 | Register Access - HSP clock 0 | 00:04 |
| 4111 | Register Access - HSP clock 1 | 00:04 |
| 4112 | Register Access - HSP clock 2 | 00:04 |
| 4113 | Register Access - HSP clock 3 | 00:04 |
| 4120 | Illegal Register Request | <:02 |
| 4130 | Register Access Parity Error | <:02 |
| 4140 | HIA Channel Interrupt | <:02 |
| 4180 | Physical Address Read - HIA buffer 0 | 00:06 |
| 4181 | Physical Address Read - HIA buffer 1 | 00:06 |
| 4182 | Physical Address Write - HIA buffer 0 | 00:09 |
| 4183 | Physical Address Write - HIA buffer 1 | 00:09 |
| 4184 | Partial Longword Transfers | 00:41 |
| 4190 | Virtual Address Read Test - 1 longword | 00:12 |
| 4191 | Virtual Address Read Test - 2 longword | 00:12 |
| 4192 | Virtual Address Read Test - 512 longword | 00:12 |
| 4193 | Virtual Address Read Test - 513 longword | 00:12 |
| 4194 | Virtual Address Read Test - 1000 longword | <:15 |
| 4195 | Virtual Address Write Test - 1 longword | 00:26 |
| 4196 | Virtual Address Write Test - 2 longword | 00:26 |
| 4197 | Virtual Address Write Test - 512 longword | 00:25 |
| 4198 | Virtual Address Write Test - 513 longword | 00:26 |
| 4199 | Virtual Address Write Test - 1000 longword | 00:26 |
| 4200 | HSP/HIA Clock Selection | 04:16 |
| 4210 | HSP Channel Functionality | 00:11 |
| 4220 | HSP ATU Parity Error Detection | <:02 |
| 4230 | HSP I/O Access Bit Verification | <:02 |
| 4240 | ATU PBUS Error Detection | 00:06 |
| 4250 | HSP Checkword Verification | <:02 |
| 4260 | No Transfer Response | <:02 |
| 4270 | Illegal Command Detection | <:02 |
| 4280 | Data Parity Error Transfer | <:02 |
| 4990 | HIA Internal Loopback - buffer 0 | <:02 |
| 4991 | HIA Internal Loopback - buffer 1 | <:02 |
| 4992 | HIA External Loopback - buffer 0 | <:02 |
| 4993 | HIA External Loopback - buffer 1 | <:02 |
| 4994 | HIA Local Slave Mode Transfer - read | 00:47 |
| 4995 | HIA Local Slave Mode Transfer - write | 03:54 |
| 4999 | CONVEX Register Compliance | <:02 |

### Subtest 4100, HIA Reset

Subtest 4100 verifies that the HSP can reset the HIA. The HSP forces the HIA main controller to its idle condition. It then sets the DMAENA bit in the HIA CONTROL register and then asserts the reset signal to the HIA. The CONTROL register is then read and the subtest verifies that the

DMAENA bit has been reset.

### Subtest 4101, HIA Voltage

Subtest 4101 measures and verifies the four voltages in the HIA chassis to ensure that they are within five percent of their nominal value. The nominal values used are plus and minus twelve volts, and plus and minus five volts.

### Subtests 4110 - 4113, Register Access

Subtests 4110 through 4113 test the HSP's asynchronous data transfer function (device register access) by pattern testing the following:

- HIA buffer memories in both 8 and 16 bit modes
- HIA parity memory in 8 bit mode
- The DMA register file in 16 bit mode
- The GLOMSK register in 16 bit mode

Uniqueness, bit, and column functionality tests are performed on each of the above. The interface clock is changed from 0 through 3 in Subtests 4110 to 4113 respectively.

### Subtest 4120, Illegal Register Request

The HSP makes illegal requests to the HIA and expects the HIA to return an illegal register response code. Subtest 4120 verifies that the error is correctly logged in the HSP BERRLOG and ADDRHI/LO registers, and also in the HIA REGERR register. The following accesses are made to force the errors:

- 16 bit access to the HIA parity memory
- 8 bit access to the DMA register file
- 8 bit access to the HIA CONTROL register
- 8 and 16 bit accesses to a nonexistent register in the HIA address space

### Subtest 4130, Register Access Parity Error

Subtest 4130 verifies that parity generation and detection is working for register accesses. The HSP makes a register access request with bad parity in the opcode, address, and data fields. For each request, the HSP verifies that the HIA sends a command parity error response and that the register request ends in a bus error.

The HSP then corrupts the parity on a longword in the HIA's buffer memory. Then a register access is made to the corrupted location and the subtest expects the register access to result in a bus error and the HIA to have sent a fatal error response to the HSP.

### Subtest 4140, HIA Channel Interrupt

The HSP resets the HIA by asserting the reset line on the HSP interface. The HSP then reads the HIA GLOINT register and clears the HIA FIR to clear any pending interrupts. The HSP then disables device interrupts in the HSP IER.

The subtest then uses the HIA FIR register to force a channel interrupt for each channel. For channel zero, the subtest verifies that the interrupt does not occur until the interrupt enable bits in the HIA CONTROL register and the HSP IER register are set. For all channels, the subtest verifies that the interrupt does not occur until the channel interrupt is enabled in the HIA IER.

### Subtests 4180 - 4183, Synchronous Data Transfers with Physical Addresses

These subtests exercise the HSP's synchronous data transfer functions without address translation. The amount of data and the addresses used in the subtests varies. These combinations test the HSP's end of logical page detection and data buffer valid flush operations. The subtests are divided into reads and writes. A data equals address pattern is used in all subtests. After each transfer, the destination area is checked to verify that all the data was transferred and that no additional data was written. All data lengths and addresses are longword aligned. Each subtest tests the data transfers with a sliding one and zero in the least significant twelve bits of the address.

### Subtest 4184, Partial Longword Transfers

Correct handling of partial longword transfers is verified by programming the HIA to transfer various transfers each with a different starting address and byte count. The addresses range from logical address 0 to logical address 16 and the byte count varies from 1 to 16. This causes all combinations of leading partial, block transfer, and trailing partial to be tested.

### Subtests 4190 - 4199, Virtual Address Read and Write

These subtests exercise the HSP's synchronous data transfer logical address translation operation. They vary in the amount of data they transfer and in the direction of the transfer. All data is longword aligned. The subtest virtual addresses are generated by sliding a one from bit 3 through bit 31. This causes the address translation logic to be checked in an incremental fashion. For all transfers, only the required ATU memory is initialized with valid data. The remaining portions are initialized with pointers to nonexistent main memory. A data equals address pattern is used for all transfers. The destination is initialized to the negation of these patterns for each transfer attempted. After each transfer, ATU memory registers 0, 1, 3, 4, 5, and 11 are checked for the expected values. The destination area is checked for valid data.

### Subtest 4200, HSP/HIA Clock Selection

Subtest 4200 tests the various data rates available for synchronous data transfers. For each possible clock combination, a 32 Kbyte transfer is initiated. After the transfer, the destination is read to verify data integrity. Reads and writes are both tested.

### Subtest 4210, HSP Channel Functionality

Subtest 4210 verifies that all HSP channels are functional by starting a 32 Kbyte transfer on each one. After the transfer, the destination is read to verify data integrity. Reads and writes are both tested.

### Subtest 4220, HSP ATU Parity Error Detection

Subtest 4220 verifies ATU parity error detection synchronous data transfers. The 68000 initializes ATU memory in a normal fashion except that register 11 is initialized with inverted parity. The HIA is programmed to perform one longword read from main memory. After the transfer, the subtest verifies that the CSR logs the error as a memory parity error during a SDR read. Additionally, the CSR is checked to ensure that the NEW_SDRS bit is set, the channel enable is reset, and that the global error bit is set.

### Subtest 4230, HSP I/O Access Bit Verification

Subtest 4230 verifies that the I/O access bit of *pte* levels 1 and 2 is processed correctly. For each *pte* level, a set of *pte*s is initialized in main memory with one valid entry and all others invalid. A one longword transfer is started from a valid logical address. The subtest verifies transfer completion without error. For PBUS reads, this verifies that a prefetch from a protected area does not produce an error response by the HSP if the data was not sent to the HIA. A second 2 longword transfer is then started. This causes the HSP to try to send data from a page without I/O access enabled. The subtest expects that the CSR will reflect an I/O protection error, that the NEW_SDRS bit is set along with the global error flag, that the *pte* level where the error occurred is properly encoded in the CSR, that the direction of the transfer is also properly saved in the CSR, and that the channel enable bit is reset. The subtest verifies both reads and writes.

### Subtest 4240, ATU PBUS Error Detection

Subtest 4240 verifies that the HSP can recover from PBUS errors received during various stages of processing. Initially, a pair of *pte*s level 2 are initialized in main memory. The first contains a pointer to a valid location in main memory, the second contains a pointer to nonexistent memory. A one longword transfer is started such that the data used comes from the valid page and the prefetch (on reads) comes from the nonexistent page. Then the subtest verifies that the transfer completes without error. For reads, the subtest verifies that the NEW_SDRS bit is set in the CSR.

The operation is restarted with a longword count that causes the data from the second page to be used. The CSR is checked to verify that it has the PBUS_BUS_ERROR bit set, that the CSR shows the error as having occurred during a data access, and that the CSR has the enable bit turned off and the NEW_SDRS bit on.

The above procedure is repeated moving the bad memory pointer from a *pte*2 to a *pte*1, *sdr*, and finally to the pointers to the *sdr*s changing the expected error condition appropriately. Both reads and writes are verified.

### Subtest 4250, HSP Checkword Verification

Subtest 4250 verifies the proper operation of the HSP checkword by causing the HIA to do a write to main memory of a single longword with an incorrect checkword. The checkword is corrupted one bit a time for each of the 64 bits of a longword. After each transfer, the ATU CSR is checked to verify that the checkword error bit is set.

### Subtest 4260, No Transfer Response

Subtest 4260 verifies the functionality of the channel enable bit by having the HIA start a transfer on a channel without the enable bit set. It also verifies that the HSP sent a no transfer response code.

### Subtest 4270, Illegal Command Detection

Subtest 4270 causes the HIA to send a partial write request for an illegal address/byte count combination. The subtest verifies that the HSP sends an invalid command response to the HIA. The CSR is checked to make sure that an error is not logged in the CSR.

### Subtest 4280, Data Parity Error Transfer

Subtest 4280 verifies that the HSP can detect data parity errors during synchronous transfers. The subtest programs the HIA to send data with inverted parity. Each of the 8 bytes in a longword is tested in succession. After each transfer, the ATU CSR is checked to ensure that the parity error was logged, the HSP is expected to have sent the HIA a block checksum parity error response code, and the IBPAR register on the HSP is read to verify that the correct byte in error was logged.

### Subtest 4990, 4991, HIA Internal Loopback

These subtests verify the integrity of the data alignment bus internal to the HIA. One of the two data buffers in the HIA is initialized with a pattern of walking '1's and walking '0's. The HIA is programmed to transfer the initialized buffer into the other HIA buffer. The destination buffer is then checked for data integrity. Subtest 4990 copies HIA buffer 0 to buffer 1. Subtest 4991 copies buffer 1 to buffer 0.

### Subtest 4992, 4993, HIA External Loopback

These subtests verify data bus integrity on the user interface. One of the two HIA data buffers is initialized to a pattern of walking '1's and walking '0's. The initialized buffer is then transferred to the other buffer and the destination is checked for data integrity. The test is repeated for all four data port sizes. For data port sizes less than 64 bits, the unused portions of the data bus are initialized to the contents of the used portion, thus, the data pattern is modified in the loopback process. Subtest 4992 copies HIA buffer 0 to buffer 1. Subtest 4993 copies buffer 1 to buffer 0.

### Subtest 4994, 4995, HIA Local Slave Mode Transfers

These subtests verify that the HIA can transfer data in a pseudo slave mode. The subtest tests all data port sizes for logical addresses from 0 to 7, for byte counts of 1 to 16, 4072 to 4089, and 8000 to 8015. After each transfer, the subtest verifies that no errors were logged and that the data was transferred correctly. Subtest 4994 tests transfers to the HIA, and Subtest 4995 tests transfers to main memory.

### Subtest 4999, CONVEX Register Compliance

Subtest 4999 verifies that the device attached to the HSP complies with the HIA standards. The register at `0xc00000` is read and its contents is verified to be one of, `0x0080`, `0x0010`, `0x0020`, `0x0020`, or `0x0030`. All valid values will cause a device interrupt test to be performed in the same fashion as Subtest 4140, Channel Interrupt Test except that the user may specify which channels the test is to be performed on by answering the correct prompt when *io4120* is invoked. If the value read from the id register at `0xc00000` is `0x0020`, `0x0040`, or `0x0080`, a read of `0xc00086` is made to verify the presence of the GLOINT register. If the value read from the id register is `0x0030`, `0x0040`, or `0x0080`, a read of a portion of the DMA channel memory is made for all channels specified by the user. Each channel will have a read made for registers at offsets of `0` through `0xe`.

## Class 5 Subtests

Class 5 tests verify the functionality of the user interface on the HIA.

- Register access to user device address space
- User device register error detection and correction
- User interface interrupts
- Data transfers to and from the user device in all data port sizes and data alignment addresses
- Data transfers in all four data transfer modes
- User interface handshake tests
- Data modulation tests
- User interface error detection and recovery
- Correct operation at all possible clock rates

The Class 5 subtests are listed in the following tables:

## Table io4120–7, Class 5 Subtests

| SUBTEST | DESCRIPTION | TIME (min:sec) |
|---|---|---|
| 5000 | User Interface Reset | <:02 |
| 5010 | FSE Ram | <:02 |
| 5020 | User Register Error | <:02 |
| 5030 | User Interrupt | <:02 |
| 5180 | HIA/FSE Small Read - 64 bit datum | <:02 |
| 5181 | HIA/FSE Small Read - 32 bit datum | 00:04 |
| 5182 | HIA/FSE Small Read - 16 bit datum | 00:09 |
| 5183 | HIA/FSE Small Read - 8 bit datum | 00:18 |
| 5184 | HIA/FSE Large Read - 64 bit datum | 00:03 |
| 5185 | HIA/FSE Large Read - 32 bit datum | 00:06 |
| 5186 | HIA/FSE Large Read - 16 bit datum | 00:14 |
| 5187 | HIA/FSE Large Read - 8 bit datum | 00:39 |
| 5190 | HIA/FSE Small Write - 64 bit datum | <:02 |
| 5191 | HIA/FSE Small Write - 32 bit datum | 00:05 |
| 5192 | HIA/FSE Small Write - 16 bit datum | 00:11 |
| 5193 | HIA/FSE Small Write - 8 bit datum | 00:24 |
| 5194 | HIA/FSE Large Write - 64 bit datum | 00:03 |
| 5195 | HIA/FSE Large Write - 32 bit datum | 00:07 |
| 5196 | HIA/FSE Large Write - 16 bit datum | 00:17 |
| 5197 | HIA/FSE Large Write - 8 bit datum | 00:42 |
| 5200 | 32k Slave Mode Read - 64 bit datum | 00:05 |
| 5201 | 32k Slave Mode Read - 32 bit datum | 00:03 |
| 5202 | 32k Slave Mode Read - 16 bit datum | 00:03 |
| 5203 | 32k Slave Mode Read - 8 bit datum | 00:03 |
| 5204 | 32k Slave Mode Write - 64 bit datum | 00:04 |
| 5205 | 32k Slave Mode Write - 32 bit datum | 00:04 |
| 5206 | 32k Slave Mode Write - 16 bit datum | 00:04 |
| 5207 | 32k Slave Mode Write - 8 bit datum | 00:03 |
| 5300 | 32k Chain Mode Read - 64 bit datum | 00:04 |
| 5301 | 32k Chain Mode Read - 32 bit datum | 00:04 |
| 5302 | 32k Chain Mode Read - 16 bit datum | 00:04 |
| 5303 | 32k Chain Mode Read - 8 bit datum | 00:03 |
| 5304 | 32k Chain Mode Write - 64 bit datum | 00:05 |
| 5305 | 32k Chain Mode Write - 32 bit datum | 00:04 |
| 5306 | 32k Chain Mode Write - 16 bit datum | 00:03 |
| 5307 | 32k Chain Mode Write - 8 bit datum | 00:03 |
| 5400 | 32k Normal Mode Transfer - 64 bit datum | 00:08 |
| 5401 | 32k Normal Mode Transfer - 32 bit datum | 00:06 |
| 5402 | 32k Normal Mode Transfer - 16 bit datum | 00:06 |
| 5403 | 32k Normal Mode Transfer - 8 bit datum | 00:06 |
| 5500 | 32k Extended Mode Transfer - 64 bit datum | 00:07 |
| 5501 | 32k Extended Mode Transfer - 32 bit datum | 00:06 |
| 5502 | 32k Extended Mode Transfer - 16 bit datum | 00:06 |
| 5503 | 32k Extended Mode Transfer - 8 bit datum | 00:06 |

**Table io4120–7, Class 5 Subtests (continued)**

| SUBTEST | DESCRIPTION | TIME (hr:min:sec) |
|---|---|---|
| 5600 | Kill Channel | 00:04 |
| 5610 | Data Modulation | 00:55 |
| 5620 | DMA Enable Bit | 00:01 |
| 5640 | Data Parity Detection | 00:05 |
| 5650 | Transfer Error from HSP | 00:20 |
| 5660 | User Read Parity Error Signal | <:02 |
| 5670 | Clock Selection | 4:03:06 |

### Subtest 5000, User Interface Reset

Subtest 5000 verifies that the HSP can reset the HIA. The HSP forces the HIA main controller to its idle condition. It then sets the DMAENA bit in the HIA CONTROL register and then asserts the reset signal to the HIA. The CONTROL register is then read and the subtest verifies that the DMAENA bit has been reset.

Subtest 5000 then reads the FSE errstat register and verifies that the power up bit is cleared. The subtest then resets the user device by clearing the HIA CONTROL register, and setting the HIA CONTROL user run bit. The FSE errstat register is then reread and the subtest expects the power up bit to be set.

### Subtest 5010, FSE Ram

Subtest 5010 consists of bit, column, and uniqueness tests for the FSE data buffer rams and the FSE command queue. The data buffer ram tests are performed twice, once using 16 bit accesses and once using 8 bit accesses. The command queue ram is only tested with 16 bit accesses.

### Subtest 5020, User Register Error

The NORESPON, BADACK, and BADERR registers on the FSE are read to force a register access error. This subtest verifies that each access results in a bus error.

The subtest then reads register addresses starting with c20001 and slides the one bit until the address is c30000. For each access, the subtest expects a bus error. After each access, the FSE LASTADRH and LASTADRL registers are read and verification is made that the address was latched correctly.

The parity on a 16 bit word in the FSE data buffer is then corrupted and the subtest verifies that reading that location results in a bus error. Each of the two parity bits in the word is checked individually.

### Subtest 5030, User Interrupt

Subtest 5030 uses the FSE interrupt register to force an interrupt condition to the HIA. When first asserted, user interrupts are masked in the HIA GLOMSK register. The subtest verifies that no interrupt is sent to the HSP and that the interrupt is logged in the HIA GLOINT register. User interrupts are then unmasked, and the subtest verifies that the HSP receives the interrupt and that it can clear the interrupt by writing to the FSE interrupt register.

### Subtest 5180 - 5187, 5190 - 5197, HIA/FSE Local Transfers

This series of subtests verify that the HIA and FSE can transfer data across the user interface. No synchronous data is transferred to or from the HSP. The subtests vary in the number of bytes transferred, the direction of the transfer, and the data port size used. The subtests listed as small transfers in the Class 5 Subtests table use byte counts of 1 to 16 inclusive. Subtests listed as large use byte counts of 4080 to 4096. For each iteration of the byte count, the subtest varies the logical address used from zero to seven inclusive in order to test all possible data alignments.

The HSP initializes the source and destination memory areas by register access. The HIA is then programmed to perform the transfer and the transfer is started. After the transfer, the HSP checks the HSP and HIA error registers and then reads the destination area to verify the data was transferred correctly.

### Subtest 5200 - 5207, Slave Mode Transfers

This series of subtests verify that the HSP/HIA/FSE can transfer data in slave mode. Each subtest transfers 32 Kbytes. The subtests vary in the direction of the transfer and in the data port size used. Each subtest varies the logical address used from zero to seven inclusive in order to test all possible data alignments.

The HSP initializes the source and destination memory areas by register access. The HSP/HIA/FSE is then programmed to perform the transfer. After the transfer, the HSP checks the HSP/HIA/FSE error registers and then reads the destination area to verify the data was transferred correctly. All transfers take place on channel zero.

### Subtest 5300 - 5307, Chain Mode Transfers

This series of subtests verify that the HSP/HIA/FSE can transfer data in chain mode. Each subtest transfers 32k bytes. The subtests vary in the direction of the transfer and in the data port size used. Each subtest varies the logical address used from zero to seven inclusive in order to test all possible data alignments.

The HSP initializes the source and destination memory areas by register access. The HSP/HIA/FSE is then programmed to perform the transfer. Each of the 16 channels is programmed to transfer one sixteenth of the data. The last channel in the chain is disabled in the HIA. The transfer is then started on the first 15 channels. After the transfer, the subtest verifies that the first 15 channels finished without any error. The enable bit on the last channel is then set, and the subtest expects the last channel to transfer its data. After the transfer, the HSP checks for errors, and then reads the destination area to verify the data was transferred correctly.

### Subtest 5400 - 5403, Normal Mode Transfers

This series of subtests verify that the HSP/HIA/FSE can transfer data in normal mode. Each subtest transfers 32 Kbytes. The subtests vary in the the data port size used. All subtests in this range are read/write subtests. Each subtest varies the logical address used from zero to seven inclusive in order to test all possible data alignments.

These subtests program the HSP/HIA/FSE to transfer data from main memory to the FSE, and then from the FSE back to a second location in main memory. The channels are programmed to transfer the data in such a manner that a read is followed by a write, then another read, and then another write. This verifies that all read/write combinations are functional.

After the transfer, the subtest verifies that the data in the FSE is correct and then it verifies that the second copy in main memory is correct.

### Subtest 5500 - 5503, Extended Mode Transfers

This series of subtests verify that the HSP/HIA/FSE can transfer data in extended mode. Each subtest transfers 32 Kbytes. The subtests vary in the data port size used. All subtests in this range are read/write subtests. Each subtest varies the logical address used from zero to seven inclusive in order to test all possible data alignments.

These subtests program the HSP/HIA/FSE to transfer data from main memory to the FSE, and then from the FSE back to a second location in main memory. The channels are programmed to transfer the data in such a a manner that a read is followed by a write, then another read, and then another write. This verifies that all read/write combinations are functional.

After the transfer, the subtest verifies that the data in the FSE is correct and then it verifies that the second copy in main memory is correct.

### Subtest 5600, Kill Channel

Subtest 5600 verifies that the HIA can process kill channel signals correctly. Kill channel processing is tested by programming transfers in all four transfer modes, and then having the FSE assert kill channel. Each transfer mode is tested for both reads and writes. After the signal has been asserted, the subtest verifies that the appropriate fields in the HIA and FSE were updated correctly. Transfers in slave and chain mode are expected to terminate the transfer on all subsequent channels after the signal is asserted. Transfers in normal and extended mode are expected to terminate only the channel on which the signal is received. Subsequent channels are expected to complete without error.

### Subtest 5610, Data Modulation

Subtest 5610 verifies that the HIA can process the read/write signals from FSE. The correct handling of the read/write handshake lines on the user interface is tested by programming the FSE to transfer data while modulating the handshake lines at varying intervals. After the transfer, the subtest verifies that the data was transferred correctly.

### Subtest 5620, DMA Enable Bit

Subtest 5620 verifies that turning off the DMA enable bit on an active channel will shut down a transfer in a graceful manner. The FSE is programmed to perform a transfer and then have it hold off sending or receiving data for many clock cycles to give the HSP a chance to turn off the DMA enable bit. The subtest then verifies that the data transferred before the hold command is correct. Both reads and writes are tested in normal and extended transfer modes.

### Subtest 5640, Data Parity Detection

Subtest 5640 verifies that HIA data parity checkers on the user interface are operational. The FSE is programmed to send the HIA data with bad parity in it. The test is made in each of the four transfer modes. For slave and chain modes, the subtest verifies that the transfer stops after the error is encountered and that the HIA resets the DMA enable bit in its control register. For normal and extended mode, the subtest verifies that only the channel where the error was made is disabled and the rest of the transfer finishes normally. For all transfer modes, the register file is examined to verify that the error is properly logged.

### Subtest 5650, Transfer Error from HSP

Subtest 5650 verifies that the HIA can process transfer error responses from the HSP. Two transfers are programmed which result in the HSP sending a *no transfer* response and a *fatal error* response. These two error responses are generated for reads and writes in each of the four transfer modes for both reads and writes. Errors in slave and chain modes are expected to terminate the transfer. Errors in normal and extended mode are expected to terminate the transfer on the channel receiving the error and continue with the remaining channels.

### Subtest 5660, User Read Parity Error Signal

Subtest 5660 verifies that the HIA can process the user read parity error signal correctly. A read transfer is programmed in which the FSE asserts the ReadParErr signal. The subtest verifies that the HIA logs the error correctly. Errors in slave and chain modes are expected to terminate the transfer. Errors in normal and extended mode are expected to terminate the transfer on the channel receiving the error and continue with the remaining channels.

### Subtest 5670, Clock Selection

Subtest 5670 performs Subtests 5200 through 5507 varying the HSP clock, HIA outclk, and HIA busclk over all possible clock combinations. It does not change the system backplane clock. Two passes through all the possible clock rates are made. The first pass forces the HIA to throttle the data, and the second pass does not.

THIS PAGE INTENTIONALLY LEFT BLANK

# io5000

# VMEbus I/O Processor Test

## Overview

The *io5000* test verifies the functionality of a specified VIOP. The test verifies the following on the VIOP(s) specified:

- 68020 on the VIOP can correctly execute instructions
- VIOP local memory is functional
- Memory protection and mapping registers are operational
- Cache memory can be written to and read from, and the associated cache control bits are functional
- Two VMEbus cable interfaces operate in loopback mode
- VIOP can boot a program from memory
- Voltages on all installed VMEbus chassis are within tolerance

## Prerequisites and Required Equipment

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

**Table io5000–1, Hardware Requirements**

| C1, C120 | C200 Series |
|---|---|
| MCU | Memory System[1] |
| MAU | CPX |
| SPU | SP2 |
| VIOP | VIOP |
| VBCU | VBCU |
| | PIA |

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

## Test Invocation

The *io5000* 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 *io5000* 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 io5000–1, Test Invocation Sequence

```
(spu)> cd /mnt/test RETURN
(spu)> sysreset RETURN
(spu)> mminit -s RETURN
(spu)> dshell RETURN
: test io5000 [-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 io5000** executes all *io5000* 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 powerup. 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 if 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 io5000–2, Alternate Test Invocation Sequence

```
(spu)> cd /mnt/test RETURN
(spu)> initall RETURN
(spu)> dshell RETURN
: test io5000 [-c [class number(s)]] [-s [subtest number(s)]] [+> filename] RETURN
```

## 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.

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 io5000–3, 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

 1: Enter VIOP numbers [0123][1]                    (1) ->
 2: Enter value for Disable_68k_cache flag [0,1]
                                                    (0) ->
 3: Enter OK, or :NN to return to question NN [OK]
                                                   (OK) ->
```

[1] The possible selections for this prompt can vary depending on the machine architecture.

At any time during the test parameter sequence, several options are available as denoted at the top of the Test Parmeter 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

? — Request help for the current prompt (if available)

^ — Return to the previous prompt

### Prompt Explanations

A description of the meaning of each prompt follows:

```
Enter VIOP numbers [0123]                    (1) ->
```

This prompt allows selection of the VIOP(s) to be scanned. For more than one VIOP, enter the numbers contiguously, i.e., 012. After entering the VIOP number(s), each VIOP entered is scanned to see if it exists. Any invalid number causes the test to reprompt for valid VIOP(s). The test executes only when all entries are valid.

```
Enter value for Disable_68k_cache flag [0,1]
                                                (0) ->
```

This prompt allows modification of the setup of the VIOP scan ring bit that can force the 68020 to run with its internal instruction cache disabled. Disabling the 68020 cache forces a memory reference for each instruction fetch.

```
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.

# Hardware Initialization Sequence

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

- SPU windows to main memory are initialized
- SPU local test variables are initialized
- The SPU verifies that the selected VIOPs are installed by scanning patterns into the VIOP LED register
- Verifies that the system has been reset since power-up

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

# Class Descriptions

The *io5000* contains the following six classes of subtests:

### Table io5000–2, *io5000* Test Classes

| CLASS | DESCRIPTION |
|---|---|
| 1 | 68020 subsystem tests |
| 2 | VIOP miscellaneous tests |
| 3 | Cache functionality tests |
| 4 | Interrupt tests |
| 5 | VBCU interface tests |
| 6 | VMEbus voltage tests |

**NOTE**

The first time one of the subtests in the 200 through 601 range is executed, the VIOP(s) under test must be loaded with the test program, which adds about 10 seconds to the test times.

## Class 1 Subtests

Class 1 subtests verify that the VIOP can correctly load a program from main memory. Class 1 tests reside in the VIOP's EPROM and communicate with the SPU via the LED and the auxiliary test result registers on the VIOP scan ring.

**NOTE**

There are restrictions on the execution sequence of these subtests:

- Subtest 100 must be the first subtest executed in this class.
- Subtests 101 and 102 may be executed in any order and may be executed in a loop until a failure occurs. If a failure occurs, Subtest 100 must be re-executed.
- After Subtest 103 is executed, Subtest 100 must be re-executed.

It is recommended that these tests be executed in sequential order.

### Table io5000–3, Class 1 Subtests

| SUBTEST | DESCRIPTION | TIME (min:sec) |
|---|---|---|
| 100 | VIOP Reset | :02 |
| 101 | VIOP Self-test | :31 |
| 102 | VIOP Initialization Command | :02 |
| 103 | VIOP Boot Command | :02 |

### Subtest 100, VIOP Reset

Subtest 100 resets the VIOP via the scan ring. This causes the VIOP to execute the initialization code in its EPROM. The initialization code saves the state of the VIOP as found on reset and verifies that the VIOP's 68020 can calculate a checksum. When verified, the VIOP calculates a checksum of the EPROM. If these tests pass, the VIOP enters a loop and waits for an EPROM command to be placed on the LED register. If this test fails, the VIOP informs the SPU by asserting a hard error.

### Subtest 101, VIOP Self-test

Subtest 101 is the VIOP's EPROM based self-test. It consists of 11 steps that verify the VIOP's ability to load a program from main memory. The SPU commands the VIOP to do this test by placing a code in the VIOP's LED registers via the scan ring. When this code is detected, the VIOP executes the steps listed below. If a failure is detected, the VIOP returns the step number and other relevant data via the LED register and auxiliary test result register in the scan ring.

**Step 1:** Pattern Test the Auxiliary Test Results Register

Step 1 verifies that the Auxiliary Test Results Register (AUXTRR) is usable for reporting failure information back to the SPU in the event of a failure of one of the later steps.

**Step 2:** Verify Instructions Needed for Memory Test

Step 2 performs the following:

- Test VIOP 68020 registers
- Check addressing modes
- Verify jump and branch instructions
- The following operations are executed:
  - Integer arithmetic tests
  - Data movement operations
  - Shift and rotate instructions
  - Bit manipulation

**Step 3:** Initial Memory Operations Test

Step 3 determines the size of memory by writing to the first word in each bank until a bus error occurs. It performs a walking '1's and '0's test on the first word after the EPROM, and on the first word of the following banks. Then, on the last 4 Kbytes of installed memory, it does a uniqueness test and a true/complement pattern test. Memory parity is checked by writing a word with inverted parity and then reading it to generate a parity error. The word is then rewritten with the correct parity.

**Step 4:** Verify Remaining 68020 Instructions

Step 4 uses the last 4 Kbytes of installed memory as a stack, and checks the 68020 instructions not tested in Subtest 100, VIOP Reset Test or in step 2 of this subtest. The *stop*, *trace*, and *reset* instructions are not checked. It also checks privilege violations and address errors.

**Step 5:** Test Remaining Memory

Step 5 is similar to step 3, but it checks memory from immediately after the EPROM to the word preceding the last 4 Kbytes of installed memory. No parity testing is done.

**Step 6:** Verify VIOP Local Memory Mapper

Step 6 starts by doing a walking '1's and '0's test on the memory protection register at `0xff8000`. For all 256 registers, it does a uniqueness test and a true/complement pattern test. The test copies the VIOP's EPROM to its local RAM, modifies the protection for installed memory to valid, read, write, and execute. It clears the registers for uninstalled memory. Memory protection is turned on, and the valid, read, write, and execute bits are verified from the supervisor state and the user state. At the end of the test, memory protection is disabled.

**Step 7:** PMAP Register Test

The RAMs that store the PMAP register data are checked by performing column functionality, uniqueness, and bit functionality tests. Bad parity detection is then checked. This is followed by a functionality test of the valid bit in the PMAP register.

**Step 8:** Not Used

Step 8 is not used in the self-test in order to be able to distinguish a failure of a given step with a VIOP that never started executing. The value of 8 is used by the SPU to command the VIOP to perform its self-test. As the VIOP progresses in its test, the value is incremented. By skipping step 8, the SPU can determine if the VIOP started execution of the self test.

**Step 9:** VIOP Cache Test

Step 9 clears the cache after disabling cache parity checking and clearing the longword/byte dirty flags and the tag flag registers. Then it reclears the tag flag registers and enables cache parity checking. Any pending cache or PBUS interrupts are cleared, and cache interrupts are enabled. The local memory copy of the EPROM is mapped in, and a checksum is performed on it. Next, the test sets the map registers to point to the last Mbyte of the PBUS. A unique pattern is written to the first 64 shortwords in each main memory 4 Kbyte window. As each word is written, the dirty bits are checked to see that they are updated appropriately. The test then clears the longword/byte dirty flags and marks all tag registers as loaded. Finally, the pattern written to each cache location is verified.

**Step 10:** VME Cable Interface Loopback Test

Step 10 resets the VMEbus cable drivers and places them in loopback mode. A local memory copy of the EPROM is mapped in through each cable interface and a checksum test is performed. Then the pattern left by the cache test is checked through each of the VIOP cache windows.

**Step 11:** PBUS Access Test

Step 11 performs the PBUS access test which reads, verifies, and echos a pattern placed in main memory by the SPU. This test verifies that the SPU and the VIOP agree on the

location of main memory and of the proper byte ordering in that memory. The first access to the PBUS made by the VIOP is to read the Population Configuration Map (PCM). This is needed to locate the test pattern in main memory.

### Subtest 102, VIOP Initialization Command

Subtest 102 determines the size of the installed local memory and then clears local memory from the word following the EPROM to the end. The test then disables cache parity checking. Subtest 102 performs the following for each of the map registers:

- Clears the dirty 'a' and 'b' bits
- Clears the corresponding 128 cache bytes
- Clears the dirty 'a' and 'b' bits again

Next, for C100 Series machines, the test enables cache parity checking, and points the first map register to the Population Configuration Map (PCM) on the MCU. The PCM is searched to locate the first 2 Mbytes of main memory. Once located, the map registers are initialized to point to the first half of this 2 Mbyte block. For C200 Series machines, the test reads the Memory Base Pointer (MBP) to determine the address of the first block of memory.

The motherboard slot number is used to index into a table in main memory. This table contains a pattern, previously initialized by the SPU, that is to be echoed by the VIOP. The pattern is the current time, in seconds, logically 'anded' with a mask of `0xffffff00`. When the VIOP echoes this pattern, the subtest ends.

### Subtest 103, VIOP Boot Command

> **NOTE**
>
> Subtest 102 must have completed successfully before executing Subtest 103. It is not possible to loop on this subtest.

Initially, Subtest 103 determines the size of local memory and copies the EPROM to the last 32 Kbytes of installed local memory. A checksum is performed on local memory to verify it. The test sets memory protection for installed local memory to valid, read, write, and execute. The memory protection registers for uninstalled local memory are cleared. The interrupt status register is then cleared, and the VIOP jumps to the local memory copy of the EPROM. Memory protection is enabled. The VIOP slot number is used as an index into a table in main memory. The table is located in the first Mbyte of installed main memory.

After the test begins, the VIOP enters a loop waiting for the SPU to place commands in main memory by the SPU. The valid commands are listed in the following table:

**Table io5000–4, Valid SPU Commands for Subtest 103**

| COMMAND | DESCRIPTION |
|---|---|
| *load* | Copies main memory to VIOP memory, a byte at a time. As each byte is moved, *load* tallies it into a checksum. After main memory is moved, it checks for parity errors. If no errors, the checksum is placed in main memory; otherwise, negation of calculated checksum goes in main memory. At completion, *load* waits for more commands |
| *reset* | Simulates a reset by loading the supervisor-stack pointer from memory location 0 and the *pc* from memory location 4. |
| *jump* | Jumps to the address specified in main memory. The *ssp* is set to the top of installed local memory. |

**NOTE**

This subtest checks only the *load* command.

## Class 2 Subtests

Class 2 subtests verify various parts of the VIOP that do not logically fall under any of the other classes. These subtests verify the next level of functionality past the EPROM based Class 1 subtests.

**Table io5000–5, Class 2 Subtests**

| SUBTEST | DESCRIPTION | TIME (min:sec) |
|---|---|---|
| 200 | PBUS Communication | :16 |
| 220 | PBUS Test-and-set | :02 |
| 221[1] | PBUS Test-and-clear | :02 |
| 230 | Line Clock Interrupt | :10 |
| 250 | VIOP Microprocessor Clock Margin | :18 |
| 251 | VIOP Cache Buffer Tag | :52 |
| 261 | Parity Checker | :01 |

[1] This subtest is only available on C200 Series machines.

### Subtest 200, PBUS Communication

After the VIOP has been booted, Subtest 200 tests the ability of the SPU to communicate with the program running on the VIOP via main memory by sending 1,000 no-op commands to the VIOP. The SPU puts a command number in the communication structure in main memory and waits for the VIOP to write into the structure indicating completion of the command. In this case the VIOP interprets the command as a no-op and immediately signals that the command has been completed.

### Subtest 220, PBUS Test-and-set

Subtest 220 verifies that all PMAP registers operate in test-and-set mode by placing each PMAP register in test-and-set mode in sequence. The PMAP register that logically follows the register under test is prepared for nontest-and-set operation and it verifies the operation.

A byte in the main memory command table is set to zero through the check window. It is read through the test window and the value returned is checked for a zero. The check window reads the byte in main memory and the value read is expected to be `0xff`. A second access through the test window verifies that the returned value is also `0xff`.

### Subtest 221, PBUS Test-and-clear

> **NOTE**
>
> This test is only available for C200 Series machines and will not execute on a C1 or C120 machine.

Subtest 221 verifies that all PMAP registers operate in test-and-clear mode, by placing each PMAP register in test-and-clear mode in sequence. The PMAP register that logically follows the register under test is prepared for nontest-and-clear operation and is used to verify the operation.

A byte in the main memory command table is set to one through the check window. It is read through the test window and the value returned is checked for a one. The check window reads the byte in main memory and the value read is expected to be zero. A second access through the test window verifies that the returned value is also zero. The operation is repeated for all windows.

### Subtest 230, Line Clock Interrupt

Subtest 230 enables the line clock interrupt on the VIOP and waits for 600 line clock interrupts to occur. The SPU UNIX time of day is checked to ensure that the 600 interrupts took 10 seconds to occur.

### Subtest 250, VIOP Microprocessor Clock Margin

Subtest 250 tests the ability of the SPU to select the clock frequency for the VIOP microprocessor. The VIOP has two clock frequencies that can be selected to run the VIOP 68020 microprocessor.

The frequencies are generated by two separate clock oscillators. The nominal clock frequency is 20 MHz, and the alternate, or margin, clock frequency is usually 22 MHz. The alternate clock frequency is derived from a socketed oscillator which is divided by two to produce the alternate clock frequency. The socketed oscillator may be replaced with any value between 25 MHz and 44 MHz.

The test first sets all selected VIOPs to run from its standard or nominal clock frequency. The VIOPs are rebooted because selecting the clock frequency requires resetting the VIOP. The VIOP is then commanded in a timed loop to calculate the frequency of the VIOP 68020 processor clock. The test uses the VIOP line clock interrupt as a time base to calculate the microprocessor clock speed. The VIOP reports to the SPU the timed count value as well as the state of the clock select bit in the VIOP Miscellaneous Diagnostic Register.

The SPU then calculates the MHz value of the VIOP microprocessor clock and prints the value on the screen. The SPU also checks the state of the reported VIOP clock state select bit against what it should be. After nominal clocks are verified, each selected VIOP is set up to run from its margin clock oscillator, and the test sequence is repeated. On exiting from this subtest, the clock selection state for each VIOP is restored to its original state.

### Subtest 251, VIOP Cache Buffer Tag

Subtest 251 verifies the VIOP cache buffer tags. Various cache accesses are performed to cause all the buffer tag RAMs to be tested for address uniqueness. As each tag is set, it is checked to verify that the correct tag is set and that no other tag is affected.

### Subtest 261, Parity Checker

Subtest 261 verifies the ability of the VIOP to force various parity errors and recover from those errors. The Miscellaneous Diagnostic Register contains the following three special control bits that allow specific parity errors to be generated:

- Force Tag Parity Error Bit
- Force Address Parity Error Bit
- Force Cache Parity Error Bit

The subtest sets the Force Tag Parity Error Bit and verifies that accessing a cache window causes a 68020 bus error and a cache error interrupt. The Cache Error Log is also checked to see that the Force Tag Parity Error Bit is set. The error is cleared and another window access is performed to verify that all the cache errors are cleared.

The same technique is used to test the Force Address Parity Error Bit. The subtest verifies that cache error is generated and the Cache Error Log register reports an address parity error.

The Force Cache Parity Error bit causes a bad PBUS header to be generated and causes a PBUS bus error. The subtest sets this Force Cache Parity Error bit, verifies that a PBUS error interrupt is generated, and checks that the PBUS Error Log reports a PBUS bus error.

## Class 3 Subtests

Class 3 subtests verify the functionality of the VIOP cache memory. The current subtests verify the cache in accelerated mode, normal mode, and in bypass mode. Also included is a test of the VIOP cache protection features with respect to VME controller accesses.

**Table io5000–6, Class 3 Subtests**

| SUBTEST | DESCRIPTION | TIME (min:sec) |
|---|---|---|
| 300 | VIOP Cache Accelerate Read | 1:48 |
| 301 | VIOP Cache Accelerate Write | 2:46 |
| 302 | VIOP Cache Bypass Read | 4:04 |
| 303 | VIOP Cache Bypass Write | 3:50 |
| 304 | VIOP Cache Protection | :30 |
| 310 | VIOP Cache Test of Dirty Bytes in Longwords | :37 |
| 311 | VIOP Cache Test of Dirty Longwords in a Buffer | 1:15 |
| 312 | VIOP Cache Test of Dirty Buffers in a Page | :36 |

### Subtest 300, VIOP Cache Accelerate Read

First, subtest 300 sets the accelerate bit in the PMAP register. Then it sets the PMAP register that logically follows the window under test to the nonaccelerate, nonbypass mode.

The VIOP slot number is used to index into a table in main memory that contains patterns and pointers to main memory, which are used to test the accelerate function. The SPU initializes the table before the accelerate test command is given to the VIOP(s). The table is accessed through the nonaccelerated PMAP register as described in the previous paragraph. The table has checksums and a VIOP identification that are checked to ensure the integrity of the data.

Using the information from the table, the test checks the PMAP window under test. Before any location in the page under test is referenced, an initial 64 byte pattern corresponding to buffer locations `0` to `3f` is initialized to a known pattern through the nonaccelerated window. The pattern used for the first byte in the page is the VIOP slot number plus the page number of the page under test, modulo 256. The test initializes each successive byte to the value of the previous byte plus 1, modulo 256.

Then a loop is entered in which each of the 4,096 bytes in the page is read and checked for valid data. Additionally, each time byte `0` of each of the two cache buffers is read, the PTAG register for the page is checked to ensure the following:

- Accelerate-on reference bit is set
- Loaded bit is set
- Buffer dirty bit is clear
- TAG value and its parity are properly set

Then, if the previous conditions are met, the test uses the nonaccelerated window to negate the current pattern in main memory. The original pattern remains in the cache. The pattern for the next 64 byte group then is initialized through the nonaccelerated window. When byte `1` of a buffer is read, the test checks the following:

- Accelerate-on reference bit is clear
- Loaded bit is set for both buffers

- TAG value for the current buffer has not changed, and the TAG value for the other buffer is properly set

### Subtest 301, VIOP Cache Accelerate Write

Subtest 301 initializes the main memory area used for the write test to the negation of the pattern to be written there. The test then uses the sum of the VIOP slot number plus the page number of the page under test, modulo 256, as the pattern for the first byte. Next, it enters a loop into which each of the 4,096 bytes in the page is written, one byte at a time. As each byte is written, the dirty bits for that longword are checked to make sure that they are set to the expected value. The test then checks the PTAG register for valid data. When byte `0x3f` of each buffer is written, the corresponding area of main memory is read through the nonaccelerated window to ensure that the pattern has not yet been written.

Every time byte 0 of a cache buffer is written, beginning with byte `0x40` in the page under test, the PTAG register is checked to verify that the accelerate-on reference bit is set. When byte 1 of a buffer is written, the nonaccelerated window reads main memory to verify that the pattern for the previous 64 byte block is written to memory and to verify that the accelerate-on reference bit resets.

After the last 64 byte block is written, the window under test is flushed so that all data is written to memory. Finally, the test again checks the pattern in the entire 4,096 byte block for integrity through the nonaccelerated window.

### Subtest 302, VIOP Cache Bypass Read

Subtest 302 checks each of the windows in the VIOP cache. As in the accelerate read test, this test places the window that logically follows the window under test in the nonaccelerate, nonbypass mode and verifies the operation of the window under test. Then it initializes the main memory area for the bypass test to the negation of the pattern to be written there. It uses the same patterns as those listed for Subtest 300, VIOP Cache Accelerate Read.

For each 4,096 bytes in the window, the test first uses the nonaccelerate window to write a 1 byte pattern to main memory. Then it reads the byte just written through the window under test and verifies the data. Finally, it negates the byte in main memory through the nonbypass window.

### Subtest 303, VIOP Cache Bypass Write

Subtest 303, the logical complement of Subtest 302, tests each window in the VIOP cache. It initializes the main memory area used to test the cache to the negation of the pattern that is to be written there. It uses the same pattern described in the cache accelerate tests.

For each of the 4,096 bytes in the window, the test first uses the window under test to write a 1-byte pattern to main memory. Then it uses the nonbypassed window to verify that the pattern was written to main memory.

### Subtest 304, VIOP Cache Protection

This subtest tests access protection of the VIOP cache. The 68020 on the VIOP simulates VME controller access into the VIOP cache by placing the cable interfaces into loopback mode and making accesses into VME address space. The VIOP cache protection bits in the PMAP registers determine which VME controllers can access a particular cache window. The subtest tests all of the available protection modes of each cache window for all possible VME controller accesses.

### Subtest 310, VIOP Cache Test of Dirty Bytes in Longwords

Subtest 310 verifies that the VIOP cache can correctly update main memory from its cache windows for all possible combinations of dirty bytes in a cache longword.

### Subtest 311, VIOP Cache Test of Dirty Longwords in a Buffer

Subtest 311 verifies that the VIOP cache can correctly update main memory from its cache windows for all possible combinations of dirty longwords in a cache buffer.

### Subtest 312, VIOP Cache Test of Dirty Buffers in a Page

Subtest 312 verifies that the VIOP cache can correctly update main memory from its cache windows for all possible combinations of dirty buffers in a cache page.

## Class 4 Subtests

Class 4 includes only the PBUS interrupt test. Other interrupts are tested in various other tests, such as the Subtest 101, VIOP Self-test.

**Table io5000–7, Class 4 Subtests**

| SUBTEST | DESCRIPTION | TIME (min:sec) |
|---|---|---|
| 400 | PBUS Interrupt | :02 |

### Subtest 400, PBUS Interrupt

Subtest 400 verifies the ability of the VIOP to request and use the PBUS interrupt bus. The subtest consists of the following three steps:

**Step 1:** Interrupt Receive Test

In the receive test, all 256 PBUS interrupts are sent by the SPU to the VIOP, one at a time. After each interrupt is sent, the VIOP verifies that only one interrupt was received and that the interrupt was received by the proper group. The group is determined by taking the modulo 4 residue of the interrupt number.

**Step 2:** Interrupt Transmit Test

In the interrupt transmit test, the VIOP sends the SPU interrupt numbers 8, 9, 10, and 11, the four interrupts that the SPU is capable of receiving. The VIOP verifies that the acknowledge for each interrupt is received and the SPU verifies the reception of the four interrupts.

**Step 3:** Interrupt Loopback Test

The interrupt loopback test causes all 256 interrupts to be sent and received by the VIOP in each of the four possible groups. Like the receive test, as each interrupt is sent, the VIOP verifies that only one interrupt is received and that the acknowledge is also received.

> **NOTE**
>
> The terms receive and transmit are from the VIOP's point of view.

## Class 5 Subtests

Class 5 subtests verify the ability of the 68020 to do accesses to the VMEbus Control Unit (VBCU). VMEbus cable interfaces, interconnecting cables, and the VBCU itself are tested for address and data line functionality. Also the VBCU-forced interrupt capability is exercised.

**Table io5000–8, Class 5 Subtests**

| SUBTEST | DESCRIPTION | TIME (min:sec) |
|---|---|---|
| 500 | VBCU Cable Pattern | :01 |
| 501 | VBCU Forced Interrupt | :01 |

### Subtest 500, VBCU Cable Pattern

Subtest 500 verifies continuity of the VBCU cable address and data lines. The VBCU is set into pattern test mode and accesses are performed to test the cable with a walking '1's and '0's pattern. The VBCU address save register is used to read back the last VME address accessed to verify the address lines.

### Subtest 501, VBCU Forced Interrupt

Subtest 501 uses the VBCU Forced Interrupt Register to test the ability of the VBCU to interrupt the 68020. The subtest forces all eight available interrupts and checks that the correct interrupt vector is received for each one for interrupt levels 1 and 3.

## Class 6 Subtests

Class 6 subtests uses the VBCU analog to digital converters to measure the VME chassis power supplies. The supplies are tested in normal mode and the +5 supply is also tested under margin conditions.

**Table io5000–9, Class 6 Subtests**

| SUBTEST | DESCRIPTION | TIME (min:sec) |
|---|---|---|
| 600 | VMEbus Voltage | :01 |
| 601 | VME Chassis Power Supply Margin | :01 |

### Subtest 600, VMEbus Voltage

Subtest 600 reads and checks the four voltages in each of the VMEbuses for out-of-tolerance conditions. The voltages and tolerances for a VMEbus are listed in the following table:

**Table io5000–10, VMEbus Voltages and Tolerances**

| VOLTAGE | MINIMUM TOLERANCE | MAXIMUM TOLERANCE |
|---|---|---|
| +12.00 | +11.40 | +12.60 |
| +05.00 | +04.75 | +05.25 |
| -12.00 | -12.60 | -11.40 |

### Subtest 601, VME Chassis Power Supply Margin

Subtest 601 performs the same voltage test as Subtest 600 on the VME chassis +5 volt supply for upper and lower voltage margins. On some VME chassis, the margin capability is disabled by a jumper on the VME backplane. In this case, the subtest prints the message:

```
Subtest 600 VME 0 not tested - margining disabled
```

# dev4100

# Multibus SMD Disk Test

## Overview

The *dev4100* test checks the Xylogics 450/451 controller and any attached Storage Module Drive (SMD) devices.

## Prerequisites and Required Equipment

This test is dependent on the existence and correctness of the SPU disk file */mnt/bin/lib/DB_diskfmt*. Refer to the section "Disk Parameters File, *DB_diskfmt* Description" in this test description for more information about this file.

The test requires the system configuration in one of the next two tables depending on the type of machine under test.

**Table dev4100–1, Hardware Requirements (C1, C120)**

| ITEM | MINIMUM | MAXIMUM |
|---|---|---|
| Xylogics 450/451 controller | 1 | 12 |
| SMD device | 1 | 12 |
| Input/output processor (IOP) | 1 | 5 |
| Multibus card cage | 1 | 10 (2/IOP) |
| Service processor unit (SPU) | 1 | 1 |
| Multibus control unit (MBCU) | 1 | 12 |
| Memory control unit (MCU) | 1 | 1 |
| Memory array unit (MAU) | 4MB | System limit |

**Table dev4100–2, Hardware Requirements (C200 Series)**

| ITEM | MINIMUM | MAXIMUM |
|---|---|---|
| Xylogics 450/451 controller | 1 | 12 |
| SMD device | 1 | 12 |
| Input/output processor (IOP) | 1 | 12 |
| Multibus card cage | 1 | 12 |
| Service processor unit (SP2) | 1 | 1 |
| Multibus control unit (MBCU) | 1 | 12 |
| Memory system (one odd, one even) | 1 | System Limit |
| Peripheral interface adapter (PIA) | 1 | 4 |
| CPU utilities (CPX) | 1 | 1 |

If a SMD is not available, only some of the Class 1 subtests can be run to check a Xylogics

450/451 controller. (Refer to Table dev4100-5, Class 1 Subtests in this chapter for the Class 1 subtests that can be executed without a SMD device.)

# Test Invocation

The *dev4100* 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 *dev4100* 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 dev4100–1, Test Invocation Sequence**

```
(spu)> cd /mnt/test RETURN
(spu)> sysreset RETURN
(spu)> mminit -s RETURN
(spu)> dshell RETURN
: test dev4100 [-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 dev4100** executes all *dev4100* 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 powerup. 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 if 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 dev4100–2, Alternate Test Invocation Sequence**

```
(spu)> cd /mnt/test RETURN
(spu)> initall RETURN
(spu)> dshell RETURN
: test dev4100 [-c [class number(s)]] [-s [subtest number(s)]] [+> filename] RETURN
```

## 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.

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

**Table dev4100–3, Getting Help During Test Parameter Entry**

| CHARACTER | DESCRIPTION |
|---|---|
| ? | Provides the help information |
| d | Displays *diskfmt* file containing the drive parameters |
| e | Displays drive entries you have made |
| h | Provides help for a specific prompt |
| i | Displays the */ioconfig* file |

After the desired help information is displayed, the system beeps and redisplays the last prompt.

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

## Figure dev4100–3, 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
?      Prints an additional help menu
 1:  Are writes to disks allowed [y,n]            (n) ->
 2:  Number of errors allowed per device each subtest
    [0-65535]                                     (0) ->
 3:  Number of devices failed after which test aborts
    [0-65535]                                     (0) ->
 4:  Run subtest 112 to check Attn/Ack [y,n]      (n) ->
 5:  Ioconfig file []                    (/ioconfig) ->
                PERIPHERAL CONFIGURATION DATA[1]
      CCU    Chassis  Type      CSR    Int  Unit  TYPE
     -------- -------  --------  -----  ---  ----  --------
 1)  iop  4     0    DKC-001   0x3f0    2     0    DKD-008
 2)  iop  4     0    DKC-001   0x3f0    2     1    DKD-005
 3)  iop  4     0    DKC-001   0x3f0    2     1    DKD-005
 4)  iop  4     0    DKC-001   0x3f0    2     0    DKD-008
 5)  iop  4     0    DKC-001   0x3f0    2     1    DKD-008
Enter device 99 to begin user-defined configurations or 0 to end selection
 6:  Device selection [0,1-5,99]                  (0) ->
 7:  Already formatted [y,n]                      (y) ->
Enter device 99 to begin user-defined configurations or 0 to end selection
 8: Device selection [0,1-5,99]                   (0) ->
Enter IOP -1 to end user-defined configurations
 9:  IOP [0,3-7]                                   () ->
10:  Multibus Chassis [0-3]                       (0) ->
11:  Controller Offset in Multibus [0x0-0xfff]
                                                (0x3f0) ->
12:  Interrupt number [0-7]                       (2) ->
13:  Unit number [0-3]                            (0) ->
14:  Drive name []                          (DKD-005) ->[2]
15:  Already formatted [y,n]                      (y) ->
Enter IOP -1 to end user-defined configurations
16:  IOP [0,3-7]                                  () ->
17:  Enter OK, or :NN to return to question NN [OK]
                                                 (OK) ->
          ********** WARNING!  ***********
        THIS TEST IS POTENTIALLY DATA DESTRUCTIVE!
        IF YOU CONTINUE, DATA COULD BE LOST!
Do you want to continue [yn]     ->
```

[1] The configuration data is only displayed once; to display it again, type **i** (RETURN) at any prompt during test parameter query.

[2] The drive name must match with a drive in the disk parameters file (*/mnt/bin/lib/DB_diskfmt*). To display the *DB_diskfmt* file, type **d** (RETURN) at any prompt.

At any time during the test parameter sequence, several options are available as denoted at the top of the Test Parmeter 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

**?** — Request help for the current prompt (if available)

**^** — Return to the previous prompt

## Prompt Explanations

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

```
Are writes to disks allowed [y,n]                    (n) ->
```

Enter **y** to allow writes to the selected disks. The test itself screens all commands before they are allowed to execute and inhibits any operations which are data destructive regardless of whether the write protect switch on the drive is set or not.

```
Number of errors allowed per device each subtest
[0-65535]                                            (0) ->
```

Enter the number of errors that can occur and still allow a device to continue. For instance, if **3** is entered, a device can have 0-3 errors and continue running. On the fourth error, the device stops and is not restarted for the duration of the test.

```
Number of devices failed after which test aborts
[0-65535]                                            (0) ->
```

Each time a unit fails (exceeds the subtest error limit), a count of the number of failed devices is incremented. When a controller fails, the count is incremented for the controller and for each active drive on the controller. If this count exceeds the number of devices specified at this prompt, the entire *dev4100* test aborts.

```
Run subtest 112 to check Attn/Ack [y,n]              (n) ->
```

> **NOTE**
>
> This feature is not used in the OS driver, so this feature does not affect the use of the controller; therefore, enter **n**. Refer to Subtest 112, Verify Attention Request/Acknowledge for more information.

Enter **y** to execute Subtest 112, Verify Attention Request/Acknowledge. Subtest 112 verifies an active controller will stop processing and acknowledge an attention request which allows the CCU driver to add or remove Input/Output Parameter Blocks (IOPBs) from an active chain. Enter **n** and Subtest 112 will not be executed.

```
Ioconfig file []                                    (/ioconfig) ->
```

This test query allows selection of a file that contains peripheral device descriptions. The default displays the system's configuration file. To display an I/O configuration file previously created, enter the filename. The configuration file is then displayed. (A relative or absolute path may be specified.)

If drives are to be tested with other than the expected configuration, a new I/O configuration file may be created. As an alternative, nonstandard configurations may be entered by responding to the user-defined configuration test parameters (questions 9-16 as shown in Figure dev4100-3, Test Parameter Menu). Then the drive and/or controller does not have to be defined in an I/O configuration file. Drives from an I/O configuration file can be selected and also drives can be defined with user-defined configurations.

```
Device selection [0, 1-5, 99]                              (0) ->
```

Enter the number of the device to be tested. Only one device can be selected with each device selection prompt. A total of 12 devices can be selected in any order.

If **99** is entered, the following four test parameters do not display. Instead, the user-defined prompts are displayed, beginning with prompt `IOP [0, 3-7]`. If **0** is entered, user selection of devices terminates and the prompt `Enter OK` is displayed.

```
Already formatted [y,n]                                    (y) ->
```

If **y** is entered the drive must be formatted prior to running *dev4100* Class 1 subtests.

```
Device selection [0, 1-5, 99]                              (0) ->
```

Enter the number of another device from the I/O configuration file. If **0** is entered, selection of devices is terminated. If **99** is entered, prompts are displayed that allow configurations to be defined that do not exist in the I/O configuration file.

```
IOP [0, 3-7]                                               () ->
```

This prompt begins the queries for user-defined drives; additional prompts are displayed requesting hardware configuration information. Enter the CCU slot number for the desired IOP or **-1** to terminate device selection. On a C1 or C120, the valid range of IOP numbers is 3-7. On a C200 Series machine, the valid range for IOP numbers is 0-3.

```
Multibus chassis [0-3]                                     (0) ->
```

Enter the number of the chassis to be tested. (The drive is attached to a controller which is in a Multibus chassis.)

```
Controller Offset in Multibus [0x0-0xfff]
                                                       (0x3f0) ->
```

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

```
Interrupt number [0-7]                                     (2) ->
```

Enter the interrupt level of the controller within the Multibus.

```
Unit number [0-3]                                    (0) ->
```

Enter the unit number of the drive to be tested. The unit number entered should also be the unit selected on the drive. (Up to four drives can be attached to one controller; each drive is assigned a unit number which is usually switch selectable on the drive.)

```
Drive name []                                  (DKD-005) ->
```

Enter the drive name. The response to this query must be in the form of DKD-xxx and the name must match one of the drive definitions in the disk parameters file (*/mnt/bin/lib/DB_diskfmt*).

```
Already formatted [y,n]                              (y) ->
```

Enter **y** if the drive has been previously formatted by the utility *diskfmt*; otherwise, enter **n**. If **y** is entered, time is saved because the subtests do not need to format the disk before starting read and write I/O operations.

```
IOP [0,3-7]                                          (0) ->
```

Enter the CCU slot number for the desired IOP to set up test parameters for another user-defined drive; otherwise, enter a **-1** or RETURN to terminate device selection. On a C1 or C120, the valid range for IOP numbers is 3-7. On a C200 Series machine, the valid range for IOP numbers is 0-3.

```
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.

```
Do you want to continue [yn]                          ->
```

Enter RETURN to begin test execution, or enter **n** to abort the *dev4100* test. 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 dev4100–4, Sample Test Parameter Summary**

```
                TEST PARAMETER SUMMARY

Are writes to disks allowed                                  :
Number of errors allowed per device each subtest             : 0
Number of devices failed after which test aborts             : 0
Run subtest 112 to check Attn/Ack                            :
Ioconfig file                                                : /ioconfig
Enter OK, or :NN to return to question NN                    : OK

                        DRIVE CONFIGURATION DATA

         IOP Mbus Mbus  Int   Unit  #    #  Phys  Log Pre-
          #   #   CSR  Level   #   Cyl  Hds  Sec  Sec Fmtd   Name
         --- --- ----- -----  --- ---  ---  ---  --- ---  -------
    1.    4   0  0x3f0   2     0  1024  27   68   67   y    DKD-008
 1000.    4   0  0x3f8   3     1   760  19   60   59   y    DKD-005

Performing sysreset of ccus and memory...
Initializing I/O subsystem and Loading IOP(s)
```

# 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.

# Class Descriptions

This test contains the following two classes of subtests which are listed in the following table:

**Table dev4100–4, *dev4100* Test Classes**

| CLASS | DESCRIPTION |
|---|---|
| 1 | Controller functionality tests |
| 2 | Drive functionality tests |

## Class 1 Subtests

Class 1 subtests exercise all the commands of the Xylogics 450/451 controller to verify its functionality. The subtests contained in *dev4100* are shown in the following table:

**Table dev4100–5, Class 1 Subtests**

| SUBTEST | DESCRIPTION | SMD REQUIRED | WRITE REQUIRED | TIME (min:sec) |
|---|---|---|---|---|
| 100 | Performs controller reset and checks default drive status for each drive | | | 0:01 |
| 101 | Executes self-test command | | | 0:01 |
| 102 | Executes DMA test command | | | 0:01 |
| 103 | Performs controller buffer write/read commands (max throttle) | | | 0:01 |
| 104 | Verifies command chaining | | | 0:01 |
| 105 | Performs NOP command; verifies that specified drives are online | x | | 0:01 |
| 106 | Executes set drive size and read drive status commands | | | 0:01 |
| 107 | Formats a track of diagnostic cylinder and reads track header; uses all interleaves | x | x | 0:02 |
| 108 | Verifies write track headers command; reads back with read track headers command | x | x | 0:01 |
| 109 | Verifies write and read (with verify) commands (all throttle settings) | x | x | 0:02 |
| 110 | Verifies seek command | x | | 0:01 |
| 111 | Performs write and read header, data, and ECC command | x | x | 0:01 |
| 112 | Verifies attention request/acknowledge | x | | 0:02 |
| 113 | Verifies that controller rejects bad heads, cylinders, sectors, and a bad command | x | x | 0:01 |

### Subtest 100, Perform Controller Reset and Read Drive Status Command

Subtest 100 performs a controller reset on each controller and verifies that the reset works by checking the address and relocation registers of the controller. It also performs a pattern test of these registers with walking '0's and '1's patterns.

### Subtest 101, Execute Self-test Command

Subtest 101 executes each controller's self-test by first using polling, then using interrupts in place of polling, and repeating the self-test.

### Subtest 102, Execute DMA Test Command

Subtest 102 executes a Direct Memory Access (DMA) test, transferring patterns that check every data line to each controller. Each pattern consists of one data bit turned on, while other bits are set to zero. The test causes the *on* bit to rotate through each data bit to verify that each bit can assume an *on* and *off* state and to check that no data bit affects other data bits.

### Subtest 103, Perform Controller Write/Read

Subtest 103 performs a 512 byte write to and read from the memory buffer in each controller to verify that the *load* and *dump* buffer commands work. During testing, the throttle setting is set to 128 words per DMA burst, which is the maximum allowed. The subtest writes and verifies a `0x55` pattern followed by a second write and verify using a `0xAA` pattern.

### Subtest 104, Verify Command Chaining

Subtest 104 verifies that command chaining works by chaining a write followed by a read of a 512 byte buffer within the controller. If other controllers are available, then this subtest executes until all controllers are checked for chaining. The test performs two write/read sequences per controller. The first write uses a repeating pattern of `0x55`; the second, a `0xAA` pattern.

### Subtest 105, Perform NOP Command

Subtest 105 performs a *NOP* command on all drives selected via the test parameter queries. The *NOP* command selects a drive, checks for drive ready, and then de-selects the drive. If any of the selected drives show up as not ready, this test fails.

### Subtest 106, Execute Set Drive Size and Read Drive Status Commands

Subtest 106 performs a *read drive status* command on all drives of every controller. It uses the *set drive size* and *read drive status* commands to verify that the configuration can be changed.

### Subtest 107, Verify Format and Read Track Headers Commands

Subtest 107 formats track 0 on the diagnostic cylinder and read verifies the success of the format by performing a *read track headers* command. The test checks this information to verify an interleave of 1 and the correct number of headers. Then the test attempts interleaves of 2, 3, 4, etc., through an interleave of 16 and verifies the interleave was formatted correctly.

### Subtest 108, Verify Write Track Headers Command

Subtest 108 verifies the *write track headers* command by performing a *read track headers* on track 1 of the diagnostic cylinder. It then writes the headers back out with all head sector numbers set to 0. The test verifies the changed headers by performing another *read track headers.*

### Subtest 109, Verify Write and Read Commands

Subtest 109 ensures that the *write* and *read* commands work. This subtest first identifies two consecutive sectors on track 2 of the diagnostic cylinder. Then it writes and read verifies the following data:

**Table dev4100–6, Subtest 109, Write and Read Data**

| SECTOR (of the two chosen) | PATTERN | SECTOR |
|---|---|---|
| 1st | 0x55 | Single sector I/O |
| 1st | 0xAA | Single sector I/O |
| Both | 0x55 | Multisector I/O |
| Both | 0xAA | Multisector I/O |

### Subtest 110, Verify Seek Command

Subtest 110 verifies the *seek* command completes without error. First it seeks to the diagnostic cylinder and confirms its position by reading track headers. A *seek* is then executed to the maximum cylinder and a *read track headers* is again used to confirm positioning.

> **NOTE**
>
> Although the *seek* command completes without error, this subtest may not detect that the controller did not seek the drive. Although a *read track headers* command is issued after the seek to verify the head is on the correct cylinder, the head may get to that cylinder through the implied seek of the *read track headers* command instead of by the *seek* command. Thus, an error in the *seek* command would not be detected. This also implies that the controller did not update its internal cylinder position when the *seek* command was issued, because, if it had, the implied seek would not occur. For these reasons, it is highly unlikely that this test would fail to detect a seek error.

### Subtest 111, Perform Write and Read Header, Data, and ECC

Subtest 111 reads header, data, and Error Correcting Code (ECC) information from each drive and writes it back. Header, data and ECC are known as HDE. Before beginning, it locates two consecutive sectors on track 3 of the diagnostic cylinder. The test writes and read verifies these sectors with test patterns 0x55 and 0xAA, respectively. These sectors are read using the *read HDE* command. It verifies the headers and data fields, and then swaps data and data ECCs between the two sectors. Next, the *write HDE* command writes the sectors back out, and then the test issues the *read HDE* command to read the data back from the two sectors. When the test completes, the data should be in the opposite sectors from where it was originally written.

### Subtest 112, Verify Attention Request/Acknowledge

Subtest 112 verifies that the attention request/acknowledge feature of the controller works. This technique is used when the controller driver wants to add parameter blocks onto an already executing chain.

> **NOTE**
>
> This feature does not work at the time of this printing because the controller has a known flaw; therefore, Subtest 112 should not be enabled. It is also not used in the OS driver. If this feature is corrected in the future, this test should be enabled.

The test builds a chain of four *read track headers* commands to each drive on the minimum, maximum, and back to the minimum cylinder to ensure they do not complete before the last two parameter blocks are added to the chain. The last parameter block performs one more minimum-maximum track seek. Once all commands for a drive have completed, all parameter blocks are checked to verify that they completed successfully.

### Subtest 113, Verify Controller Reject

Subtest 113 first sends a head address outside the limit the controller is checking; the controller should reject it. Then the test sends a cylinder address outside the drive's range and a sector address outside the controller's limits. The controller should reject these as well.

## Class 2 Subtests

Class 2 contains subtests that verify the drive's functionality. These tests verify that the drive can seek, switch heads, read, and reject a bad head or cylinder. The subtests contained in *dev4100* are shown in the following table.

**Table dev4100–7, Class 2 Subtests**

| SUBTEST | DESCRIPTION | SMD REQUIRED | WRITE REQUIRED | TIME (min:sec) |
|---|---|---|---|---|
| 200 | Verify head switching | x | x | 0:02 |
| 201 | Verify sequential-track seeks | x | x | 0:45 |
| 202 | Perform minimum to maximum track seeks | x | | 0:10 |
| 203 | Perform accordion seeks | x | | 1:15 |
| 204 | Perform random seeks | x | | 1:15 |
| 205 | Verify detect of forced faults (bad cylinder, head, and sector) | x | | 0:01 |

### Subtest 200, Verify Head Switching

Subtest 200 verifies that the heads switch correctly. The test formats all heads on the diagnostic track and then reads them back with a *read track headers* command. It checks the head number in a header on each head and verifies the shift of the headers by one sector per head.

### Subtest 201, Verify Sequential Track Seeks

Subtest 201, performs a *read track headers* command on each cylinder to confirm positioning.

### Subtest 202, Perform Minimum to Maximum Track Seeks

Subtest 202 performs a *min-to-max* track alternate seek. Special formatting takes places for Class 2 subtests, if necessary, before the main portion of this subtest executes. The seek begins with an implied seek to the minimum track with a *read track headers* command. After the test verifies the cylinder number, a seek to the maximum track is performed using the *read track headers* command. When this cylinder is verified, the seek repeats to minimum and maximum tracks 100 more times with verification at each track. Each seek is an implied seek using the *read track headers* command. The seek then repeats, using the *seek* command with no verification until the seek is complete. To complete the test, the last track is read with a *read track headers* command and positioning is verified.

### Subtest 203, Perform Accordion Seeks

Subtest 203 exercises the drive in an accordion seek. The seek starts with a *seek* to the minimum track, maximum track, min+1, max-1, min-2, max-2, etc., until a one-track seek is performed. The *read track headers* command actually issues an implied seek for each seek. The test verifies the seek by checking one of the headers just read. The accordion seek then repeats using the *seek* command and no verification. After the last seek is performed, a *read track headers* command is performed to verify that the final positioning is correct. The test chains commands for every drive, if allowed.

### Subtest 204, Perform Random Seeks

Subtest 204 performs 1000 random seeks within the specified minimum and maximum cylinder. Random numbers are created by initializing C's random number generator with a seed value of 113 and then by calling *rand* for successive numbers. All seeks are implied by the *read track headers* command, which verifies positioning. The test then performs the random seek using the *seek* command with no verification until the seeks are complete. Finally, the test verifies the last cylinder with a *read track headers* command. The test chains commands for every drive, if allowed.

### Subtest 205, Verify Detect of Forced Faults

Subtest 205 sends a *set drive size command,* which sets the head and cylinder limits to two greater than the maximum values specified in the file */mnt/bin/lib/DB_diskfmt*. Then the test commands the controller to read a track from a cylinder that is two greater than the maximum. This should result in a seek error. No error may occur if the number of cylinders specified in

*/mnt/bin/lib/DB_diskfmt* is less than the actual number of cylinders. In this case, an `Expected error did not occur` error is displayed.

Next, the test sets the head to maximum plus 2 and selects the diagnostic cylinder. This generates a write fault. After the write fault, the error does not clear until a drive reset is performed. To verify this, the test performs a read of head 0 of the diagnostic cylinder. A write fault should still occur. Finally, the test resets the drive and reads head 0 again; something other than a write fault should occur.

> **NOTE**
>
> A CDC 9766 drive will fail this test because it reports no error when an attempt to seek two cylinders beyond the defined maximum is made.

# Disk Parameters File, *DB_diskfmt* Description

The disk parameters file, */mnt/bin/lib/DB_diskfmt*, contains information about disk drives. This information is required to be able to format/test new drives. Each line in the file contains a number of fields separated by one or more spaces. Comments start with a # in column 1. To add new drives, create a new entry with all the fields defined, and then add the drive to the */ioconfig* file. As of the time of this printing, *DB_diskfmt* contains the following information for all CONVEX machines:

## Figure dev4100–5, Contents of the *DB_diskfmt* File

```
# DB_diskfmt - file of disk parameters
# >>>> WARNING - DO NOT USE 'diskfmt' TO FORMAT! It is no longer compatible
# >>>>           with the CONVEX FORMAT! Instead, format MBUS-attached drives
# >>>>           with 'dev4110' and VME-attached drives with 'dev5130'.
# KEY FOR DRIVE NAMES (unformatted capacity is given in parentheses):
#  Name          Description           Name          Description
# DKD-001       Fujitsu Eagle (452MB) DKD-008,208   NEC 2363 (1080MB)
# DKD-002       CDC 9766 (300MB)      DKD-214       Hitachi DK514-38 (356MB)
# DKD-005,206   NEC 2352 (500MB)
#------- XYLOGICS 450/451 SMD CONTROLLER (MBUS) -------
#  a     b    c   d  e  f    g     h    i j k  l  m   n    o  p
DKD-001 2   842 20 46 45 4800 28160 1 0 1  0  0  smd mfm y
DKD-002 0   823 19 32 31 5040 20160 1 0 1  0  0  smd mfm y
DKD-005 0   760 19 60 59 4832 36288 1 0 1  0  0  smd 2-7 y
DKD-008 1  1024 27 68 67 4816 40960 1 0 1  0  0  smd 2-7 y
#------- INTERPHASE 4201 ESDI CONTROLLER (VME) --------
#  a     b    c   d  e  f    g     h    i j k  l  m   n    o  p
DKD-214 0   903 14 51 50 4736 30240 5 5 1  8  8 esdi 2-7 n
#------- INTERPHASE 4200 SMD CONTROLLER (VME) ---------
#  a     b    c   d  e  f    g     h    i j k  l  m   n    o  p
DKD-206 0   760 19 60 59 4832 36288 5 4 1 12 12  smd 2-7 n
DKD-208 0  1024 27 68 67 4816 40960 5 6 1 16 12  smd 2-7 n
# LEGEND:
# a - drive name                Must be DKD-0XX for Multibus and DKD-2XX for
#                               VMEbus
# b - disk type                 For Xylogics controller
# c - # of cylinders
# d - # of heads
# e - # of physical sectors     Number of actual sectors excluding runt
# f - # of logical sectors      Number of physical sectors (e) minus number of
#                               spares
# g - bits per sector           Number of bits between sector pulses
# h - bytes per track           Total number of unformatted bytes per track
# i - skew                      Sector offset from one head to the next
#                               Must be 1 when using Xylogics 450/451 cntlr
# j - # of relocation tracks    .5% of number of cylinders (c). Raise
#                               fractional part to next higher whole number.
#                               Ignored by dev4110 (Multibus formatter)
# k - interleave                sector separation between consecutive sectors
#                               Currently must be 1 for Xylogics 450/451 cntlr
# l - gap 1 size                Number of halfwords in gap before header
#                               (2 bytes per halfword)
#                               Ignored by dev4110 (Multibus formatter)
# m - gap 2 size                Number of halfwords in gap following header
#                               (2 bytes per halfword)
#                               Ignored by dev4110 (Multibus formatter)
# n - drive interface           Used to determine how to read manufacturer's
#                               defect map.  Currently, smd or esdi
#                               Ignored by dev4110 (Multibus formatter)
# o - data encoding scheme      Way data is encoded on the media. Used to
#                               select patterns for pattern test.
#                               Currently mfm, 2-7 or 1-7
# p - Are spares interleaved    For Xylogics, 'y'. For Interphase, 'n'.
```

# Diagnostic Cylinder Description

The diagnostic cylinder is located on the first full cylinder preceding the sector forwarding area. Cylinder 840 is the diagnostic cylinder on a Fujitsu Eagle drive, and cylinder 820 is the diagnostic cylinder on a CDC SMD drive.

Each track of the cylinder is independent of the other tracks since there is a "table of contents" stored at sector 0 of each track This table describes what is stored on the remainder of that track. The format of this table is shown below:

**Figure dev4100–6, Diagnostic Cylinder Table of Contents Format**

```
/* ------------------------------------------------------------------ *
 *              Diagnostic Cylinder Definitions                       *
 * ------------------------------------------------------------------ */
#define NOT_USED        0x00000000      /* position in tbl not used     */
#define NO_HDR          0x00000001      /* this sectors header deleted  */
#define TBL_CONT        0x00000002      /* sector contains tbl of cntnts*/
#define ECC_ERR         0x00000003      /* sector written with bad ecc  */
#define PATTERN         0x00000004      /* sector written with pattern  */

 struct   tbl_cont {
    int   magic_nbr;            /* identifies this as a table of contents */
    int   version;              /* version number of this table */
    int   cyltksc;              /* sector header for this sector*/
    struct {
          int   sec_cont;         /* defines contents of corresponding sector */
          int   pattern;          /* pattern or mask used to verify contents  */
   } sec_tbl[62];
    int   checksum;             /* arithmetic tally of all the above fields */
 };
```

Each track on the cylinder is formatted in such a way that sector 0 is always on the track; sector 0 is written and verified without error when the drive is formatted.

The first field in the table contains the magic number 0x84101500. If this number is not present in the first position of a table, then someone has altered the diagnostic cylinder.

The second field in the table contains the version number. This number is an ordinal number starting with one and increasing as needed. Its purpose is to allow some measure of compatibility of newer diagnostic cylinders with older software. This number is checked if an unknown entry in the sector table (sec_tbl) is encountered. If the version number in the table is greater than the version number of the software, the diagnostic cylinder is assumed to have been formatted with a newer revision of software, and the opcode in the sector table is new to the older software. Therefore, the software is not familiar with the opcode, and the entry in the sec_tbl is ignored.

The next field in the table contains a copy of the sector header (cyltksc) for the table of contents.

The next field in the table contains the sector table (sec_tbl). This table describes what is stored on the rest of the sectors in this track. The sector table consists of 62 entries of 2 int's. Each entry (0 - 61) in this table, corresponds to a sector (0 - 61) on the track. The first position in each entry is the sector contents (sec_cont) field. If the track has more than 62 sectors, the extra

sectors are not used or checked by current software. The following table describes what is currently defined:

**Table dev4100–8, Defined Values for Sector Contents Field**

| FIELD | DESCRIPTION |
|---|---|
| NOT_USED | This sector has not been initialized and should not be checked. |
| NO_HDR | When this sector is read, the controller should return a 0x05 sector not found error. |
| TBL_CONT | This sector contains a copy of the table of contents. |
| ECC_ERROR | This sector contains a copy of the table of contents that has been corrupted to give some form of ECC error. The pattern field contains a mask which may be xor'ed with the magic number field to correct the error. If the mask contains 11 or fewer one bits, the controller should report a soft ECC error when this sector is read. If the mask contains more than 11 one bits, the error reported should be a hard ECC error. |
| PATTERN | This sector has been filled with the data contained in the pattern field. No errors should be encountered when the sector is read. |

The last field in the table of contents is a checksum. This is an arithmetic tally of all the bytes in the table of contents. This value is checked before any of the data in the table of contents is used.

# Error Codes

The *dev4100* test reports controller and device errors in a standard format. However, if additional data is available at the time of the error, it reports that data also. The basic format for the errors is:

dev4100 ERROR (Err) ccu/mb/csr/d=c/m/rrr/d Err_desc

where:

(*Err*) Identifies the error code associated with the description.

*Err_desc* Describes the type of error that occurred.

*c* Gives the CCU slot number for the IOP.

*m* Is the Multibus chassis number.

*rrr* Gives the control and status register address (CRS), which identifies the board.

*d* Specifies the device number. Each device begins at 0 with additional drives of the same type being numbered 1, 2, and 3. The number 4

indicates the controller itself was under test.

If no device is associated with the error, then the controller and device information does not appear in the error message.

Additional data may also be reported on a second line (depending on the type of error). For example, the format for data compare errors is:

Cyl:dddd Hd:dd

The following format appears when errors involve *write*, *read*, *read* and *write HDE*, *read* and *write track headers*, or *seek* commands (depending on the error, the sector number or number of seeks may not appear in the error message):

Cyl:dddd Hd:dd Sect:ddd Disp:ddd Exp:hhhh Act:hhhh #Seeks:d

where:

| | |
|---|---|
| *Cyl:dddd* | Specifies the position of heads on the drive. |
| *Hd:dd* | Identifies the head selected. |
| *Disp:ddd* | Identifies number of bytes from the beginning of the sector (data compare errors only). |
| *Exp:hhhh* | Specifies the expected value (data compares errors only). |
| *Act:hhhh* | Specifies the actual value (data compares errors only). |
| *Sect:ddd* | Indicates the sector where the error occurred. |
| *# Seeks:d* | Identifies the number of seeks executed. |

# Error Messages

During execution, if the error limit is exceeded, the test aborts and displays:

```
Subtest Termination in Progress
```

For normal test completion, a message displays in the following format:

```
************ Test started Tue Mar 11 10:30:25 1986
************ Test ended   Tue Mar 11 10:34:15 1986

dev4100 Termination complete.
```

During execution, *dev4100* outputs error messages from the controller, IOP, disk device sequencer, and device subtests. The following sections describe these errors.

## Controller Error Messages

> **NOTE**
>
> In the following error messages, IOPB represents Input Output Parameter Block.

**0x00 IOPB completed without error**
The controller completed the command successfully without an error.

**0x01 Interrupt pending**
The controller attempted an operation with a previous interrupt still pending. While an interrupt is pending, only the following operations are permitted: interrupt reset, update IOPB, controller reset, or error reset.

**0x03 Attempt to write to reg. while busy**
The IOP attempted a register write while controller busy is set. When the controller is busy, only bits 2 and 4 of the control and status register (CSR) have write access.

**0x04 IOPB not completed within two seconds**
The controller did not complete the input/output parameter block within two seconds.

**0x05 Header not found**
The controller cannot locate the requested sector. Any one of the following could cause this error:

- The actual number of physical sectors in the drive exceeds the maximum number of sectors plus 5. For example, if the controller searches 37 (32 + 5) sectors and the drive has 47 actual sectors, the controller may not compare 10 sectors for valid headers. (The controller compares headers for the maximum number of sectors plus 5.)
- The header the controller finds does not match the header ECC.
- The requested drive type and the drive type contained in the header do not match.
- A media defect may be in the header area.

**0x06 Hard ECC error**
This error only occurs on a *read* command when the controller detects a data error in the data field longer than 11 bits or when the ECC mode is disabled.

**0x07 Bad cylinder from host**
The IOP specified a cylinder address greater than the maximum cylinder number allowed.

**0x0a Bad sector from host**
This message indicates that the IOP specified a sector address greater than the maximum sector number allowed. Verify the maximum sector parameter for this drive type; retry the IOP operation.

**0x0d Runt sector too small to write header**
Either the last sector or all the sectors are too small to write a complete header. Verify the drive sector switches.

**0x0e Memory failed to respond to DMA**

This message means that the memory address by the controller failed to respond. The microprocessor provides a 10 millisecond timeout for the DMA sequencer to perform up to 128 transfers. When the timer interrupts, the controller verifies whether it is a bus master. A slave acknowledge error occurs if it is a bus master; a disk sequencer errors occurs when it is not a bus master. Verify the memory address or memory itself; retry the operation.

**0x12 Header cylinder or head is incorrect**

The cylinder or head address read from the disk does not match the IOPB cylinder and head address bytes.

This error occurs when the disk drive fails to seek to the correct cylinder. Another condition that causes this error is a corrupted disk format. Reformat the disk and retry the operation. In addition, if the head byte written on the disk does not match the selected head address, this error occurs. This indicates that there is a bad format or hardware problem.

**0x13 Seek error was corrected by cntlr**

The controller encountered a *seek* error. When the controller encounters a *seek* error, it automatically recalibrates the disk drive, clears the error, and completes the *seek*.

**0x14 Write-protected disk**

The controller attempted a *write* operation on a drive that is write-protected. First remove the write-protect and then retry the *write* operation.

**0x16 Drive is not ready or faulted**

This error indicates that the selected drive is not ready or possibly faulted. Any one of the following conditions can cause the problem:

- ALCO signal on the multibus backplane connector is low.
- Bad or improperly connected cable.
- Drive not up-to-speed or hardware error.
- Dual port access many not have been granted.
- No drive of the specified unit number is connected to the controller.

**0x17 Bad sector count of 0 from host**

This message indicates that the IOP issued the controller an IOPB with a sector count of zero. All data transfer operations require a positive sector count.

**0x18 Drive reported a fault condition**

This error messages indicates that a fault exists in the selected drive.

**0x19 Bad sector size from host**

The controller cannot write the header and data fields because the drive sectoring does not allow enough room. Any of the following conditions can cause this error to occur:

- The runt sector is too small.
- The drive contains more sectors than the number of specified data sectors plus five.
- Although the last sector is too small to be a data sector, it is included in the specified maximum sector. Either adjust the drive to include more sectors or the drive type (*DB_diskfmt*) to include fewer sectors.

**0x1a Self-test A failed**
Either the microprocessor or its internal RAM failed diagnostics.

**0x1b Self-test B failed**
This message indicates that either the microprocessor or header shift register failed diagnostics.

**0x1c Self-test C failed**
The buffer RAM failed diagnostics.

**0x1e Correctable ECC error**
The controller detected a correctable 11-bit (or less) error in the data field of the correct sector during a *read* in ECC mode 0.

**0x1f Illegal ECC Mode used by Cntlr**
During the transfer, the controller corrected one or more ECC errors in ECC mode 2, which should never be used as it does not work correctly in our configuration.

**0x20 Bad head from host**
The IOP specified a head address greater than the maximum head address allowed, which indicates a malfunctioning controller.

**0x21 Disk sequencer error**
This message indicates that the disk sequencer did not finish its operation within the allotted time. This can be caused by any of the following factors:

- The controller did not receive the servo clock signal from the selected drive. Check the cable connection.
- The controller is not receiving any read data from the selected drive. Check the cable.
- The Multibus may be preventing the controller from gaining proper access to memory.

**0x25 Seek error**
The IOP specified a cylinder address greater than the drive maximum or specified a head beyond that supported by the drive. Verify the drive parameters for the drive type tested.

**0x3f Dual port drive already connected**
The controller attempted to select a dual port drive which was connected to another controller. Since dual part drives are not supported, this error should never occur.

## Device Subtest Error Messages

**0x40 Bad block table is full**
The bad block table can hold a maximum of 639 sectors. It is extremely unlikely that this table would ever fill up so it is much more likely that the bad block table is bad. In other words, the count of the number of used entries that is stored in the table is bad. This error sometimes occurs when a disk has been only partially formatted (headers are there and pattern-test data but no bad block table exists).

**0x41 Bad sector not in Bad Block Table**

A header not found occurred on a track which usually means the sector has been relocated. However, the sector was not found in the Bad Block Table. This may be due to a disk being only partially formatted.

**0x42 No consecutive sectors found**

The HDE subtest requires that two consecutive sectors exist on head 3 of the diagnostic cylinder. This error occurs if these headers are not found.

**0x47 Bad parm to calc_cyl**

This indicates a bug in the software. Please report it.

**0x48 Bad parm to calc_hd**

This indicates a bug in the software. Please report it.

**0x49 Bad parm to calc_sec**

This indicates a bug in the software. Please report it.

**0x4a IOP print utility memory error**

The print utility, prtlog_init, was unable to allocate main memory needed for prints from the CCU. Try running **mminit** and then retrying the test.

**0x4b Device FAILED**

The specified device failed this subtest. No more testing will be performed on this device.

**0x4c Input parms don't allow write**

The specified drive does not allow writes, and a write subtest was selected. The write is inhibited.

**0x4d All copies of BBT are bad**

There are five copies of the Bad Block Table on the last cylinder of each system-formatted disk. This error occurs when none of them are readable.

**0x4e Diag. trk has all bad headers**

During creation of the diagnostic cylinder, one of the tracks was found to be unusable because none of the sectors were readable. This error is very serious because it indicates either a major media flaw or a failure of that head to read data.

**0x4f Diag. cylinder bad**

During test of the diagnostic cylinder, sector zero of each track is read and two integer fields in the sector's data are checked against expected values. If they differ, this error results. This is usually caused by *dev4100* being run followed at some time later by a run of Subtest 103 in *dev4110*. The *dev4100* test writes over the diagnostics data written by Subtest 101 in *dev4110*. This problem is easily fixed by rerunning Subtest 101 in *dev4110*. Then use Subtest 103 in *dev4110* to verify the diagnostic cylinder is okay. If it still fails, then there is a problem writing or reading sectors.

**0x50 Bad data compare on diag. cyl**

Each track of the diagnostic cylinder has most of the sectors written with known patterns. These sectors are read and a data compare is performed to verify no data degradation or overwrite of this data has occurred. If any of these sectors do not successfully data compare, this error results. This could be caused by running *dev4100* prior to running Subtest 103 of *dev4110*.

**0x51 N0 err. Hdr-not-found expected**
One sector on each of the tracks of the diagnostic cylinder is deliberately removed so that a header not found error will result. If a header not found does not occur, this error is reported. This could be caused by running *dev4100* prior to running Subtest 103 of *dev4110*.

**0x52 Soft ECC error did not occur**
Subtest 103 of *dev4110* reports this error. Each track of the diagnostics cylinder has 11 sectors with soft ECC errors written to them. If any of these 11 sectors on each track do not cause the controller to report a soft ECC error, this error is reported.

**0x53 Hard ECC error did not occur**
Subtest 103 of *dev4110* reports this error. Each track of the diagnostics cylinder has one sector that has data with 12 consecutive bits in error. When this sector is read, it should always cause the controller to report a hard ECC error. If this error does not occur, this error is reported.

**0x54 Data compare err on diag cyl**
Most sectors on the diagnostic cylinder are written with known data patterns. During the test of the diagnostic cylinder, each sector containing known data is compared. This error occurs if a data compare fails. It may indicate degraded media or a bad head.

## IOP Error Messages

**0x80 Data compare error**
Data read by the IOP did not match the expected data. The IOP performs the following data compares:
**0x81 Seek requested with a seek count = 0**
SEEK_CNT_ZERO – This indicates a bug in the software. Please report it.

**0x82 Format of partial tracks not allowed**
FORMAT_SETUP_BAD – This indicates a bug in the software. Please report it.

**0x83 No attn acknowledge after attn request**
NO_ATTN_ACK – This results when the controller is started on a chain of commands and then an attempt to get the attention of the controller in the middle of execution of the chain fails. This is a known problem with the controller and causes Subtest 112 to fail.

**0x84 MBS error attempting to return msg**
The IOP was attempting to send a message to the SPU when the MBS I/O subsystem reported an error condition.

**0x85 Desired IOPB not in active chain**
IOPB_NOT_IN_CHAIN – This is possibly due to a bad controller overwriting the pointer to the next IOPB in a chain. However, any component in the path of data from a controller to main memory is suspect.

**0x86 Message received on wrong subqueue**
The IOP received a message on a subqueue other than seven.

**0x87 Wrong subqueue active and locked**
The IOP detects a message on a subqueue other than seven, but the MBS locked the subqueue when a receive was attempted.

**0x88 Wrong subqueue active and MBS error**

The IOP detects a message on a subqueue other than seven. However, MBS reports an error when a message receive is attempted.

**0x89 Wrong subqueue active, error invalid**

The IOP detects a message on a subqueue other than seven. However, MBS reports an invalid error code when a message receive is attempted.

**0x8a Driver executed for no reason**

The Event Governed Operating System (EGOS) brought the IOP driver into execution; the driver found no reason for this action.

**0x8b Unexpected soft error from Cntlr**

The controller reported one of two soft errors when no soft error was expected. The error is one of the following:

| | |
|---|---|
| Seek Retry Required | The controller detected a seek error and automatically recovered from the error. Because the bit that allows automatic recovery is not enabled, this error should never occur. |
| Soft ECC Error Recovered | The controller thinks it automatically corrected an ECC error of eleven bits or less. Because CONVEX does not use this function, this error should never occur. |

**0x8c Cntlr busy after hard error**

After a hard error occurs, the controller goes "not busy" (according to the *Xylogics User's Manual*). In this case, a hard error was detected, and the controller was still busy when a retry was attempted.

**0x8d Retry in progress-no active chain**

NO_ACTIVE_CHAIN – This indicates a bug in the software. Please report it.

**0x8e No attn acknowledge after attn request**

Subtest 112 of *dev4100* verifies that, when a chain is actively executing, an attention request to the controller will cause the controller to halt without completing the chain and respond with attention acknowledge. This error indicates that this logic in the controller does not work properly. The Xylogics controller has this problem and will generate this error if Subtest 112 is enabled to run.

**0x8f Cntlr not busy after release attn request**

The controller is designed so that if busy is set when the attention request is asserted, busy remains set after the release of the attention request bit. If busy goes away, the controller does not meet specifications.

**0x90 Cntlr busy after release attn**
CNTLR_BUSY_ERR – The controller went not busy when during an attention request and before the controller was restarted. This is okay. However, when the controller's attention was released, the controller was found to be busy. This indicates a bad controller.

**0x91 Interrupt occurred—no IOPBs done**
The controller sent an interrupt; when the active IOPB chain was scanned, no completed IOPBs were found. This interrupt is not caused by an attention request, which is handled separately.

**0x92 EGOS routine wndw_alloc returned 0**
An attempt to set up a window to main memory from the IOP failed.

**0x93 Task init attempt-error pending**
RESPONSE_NEEDED – This indicates a bug in the software. Please report it.

**0x94 No cntlr interrupt in 5 seconds**
The controller must interrupt the driver within 5 seconds after an IOPB chain is started or a timeout occurs.

**0x96 Write/Read error in controller regs**
Write/Read test of controller registers failed.

**0x97 Cntlr was busy after drive fault**
The controller must go not busy after a drive fault. This is checked to ensure the controller is working as specified.

**0x98 Cntlr stayed busy after drive reset**
The controller is specified as going not busy after a drive reset. If it stays busy, this error is reported.

**0x99 Attempting to poll-no active IOPBs**
NOTHING_TO_POLL – This indicates a bug in the software. Please report it.

**0x9a Finished task but int. state wrong**
This indicates a bug in the software. Please report it.

**0x9b Controller is not present**
An attempt to write to the controller resulted in a BUS error, which implies that no controller is present or the controller is dead.

**0x9c Sector interleave or skew error**
The track headers were read to verify the format, and the headers were not properly interleaved or skewed.

**0x9d Bad cylinder address after seek**
A check of headers after *seek* revealed that the headers are on the wrong cylinder.

**0x9e Set drive size - bad head or sector**
A *read drive status* after setting the drive size in the controller showed that either the head or sector size was incorrectly set.

**0x9f Set drive size - bad cylinder**
A *read drive status* after setting the drive size in the controller showed that the cylinder size was incorrectly set.

**0xa0 Expected error did not occur**
Subtest 205 of *dev4100* selects a non-existent cylinder, then a nonexistent head is selected. An expected error had better occur or this error results.

**0xa1 Error bit(s) set in cntlr csr**
If a single or double error is reported in the command and status register and no other error is indicated, this error is reported.

**0xa2 No free IOPBs for retry**
NO_FREE_IOPBS – This indicates a bug in the software. Please report it.

## Device Sequencer Error Messages

**0xc0 MBS error-when SPU receiving a message**
The MBS reported an error in attempting to provide the SPU with a message.

**0xc1 Timeout because queue locked**
The SPU received a message, but all ten attempts to read the message resulted in the error *MBS queue locked* being reported by the MBS.

**0xc2 Msg interrupt but no msg found**
The SPU attempted ten times to receive a message when an interrupt occurred, but all attempts failed with a *no message* error from the MBS.

**0xc3 Bad return code from msg_build**
BAD_BUILD_RTN – This indicates a bug in the software. Please report it.

**0xc4 Timeout awaiting msg**
SPU expected the IOP to send a message, but it did not receive a message. This error message indicates that the controller failed to generate an interrupt when done, or a break exists in the path of the interrupt signal, which prevents it from being detected.

**0xc5 Rtn msgs from subq other than 7**
The SPU received a message to the wrong subqueue; all messages are expected to arrive on subqueue seven.

**0xc6 MBS returned bad error code**
A call to an MBS routine returned an error code that was unrecognizable.

**0xc7 Block transfer routine failed**
A memory to memory transfer failed. Memory may be local or main.

**0xc8 Too many devices failed. Failed subtest**
This message indicates that the test is terminating because the device failure limit was exceeded.

**0xc9 Return msg not of valid type**
The SPU received a message, but it was not a device result or system error message.

**0xca IOP's queue locked too long**
Ten attempts to send a message to the IOP failed, with a locked condition being reported by MBS.

**0xcb No room on SPU side for msg**
Ten attempts to send a message failed because no messages were available for MBS to use.

**0xcc Return msg error**
An attempt to send a message from the SPU to the IOP resulted in an MBS error being reported.

**0xcd Error while configuring I/O**
The IOP returned a configuration message that specified a controller that does not exist.

**0xce # active dev on cntlr < 0**
BAD_DEV_CNT – This indicates a bug in the software. Please report it.

**0xcf Too many controllers specified**
The user attempted to select drives on too many controllers. This test's limit is 6. The limit for *dev4110* is 12.

**0xd0 IOP echoed msgs when none sent**
The SPU received an echo message, but no message had been sent.

**0xd1 Bad cntlr # in returned msg**
BAD_MSG – This indicates a bug in the software. Please report it.

**0xd2 Outstanding msg cnt greater than 0**
BAD_MSG_CNT – This indicates a bug in the software. Please report it.

**0xd3 No msg err when sending to SPU**
A message buffer was requested from MBS on the IOP. No message buffer was available. Try again later.

**0xd4 MBS error when sending to SPU**
The IOP MBS logic reported back an error condition (catchall for unexpected errors) while attempting to send a message to the SPU from the IOP.

**0xd5 Inv. MBS err when sending to SPU**
MBS on the IOP reported a non-zero error code that is not defined.

**0xd6 MBS locked when sending to SPU**
MBS on the IOP will not allow the sending of messages because the semaphore on the SPU process's queue is not being released.

**0xd7 IOP driver device is bad**
This indicates a bug in the software. Please report it.

**0xd8 MBS error during IOP msg recv**
The MBS routine that receives messages on the IOP reported that it detected an error. Usually this means that main memory no longer contains valid message queues.

**0xd9 MBS queue locked during receive**
The SPU attempted to receive a message but its queue was locked too long. Should never happen.

**0xda MBS no message available**
A message should have been in the queue but when it was checked, there were no messages.

**0xdd Processor queue setup failed**
PQ_SYSTEM_FAIL – The SPU utility, **pqutil** was unable to initialize the processor queues which are needed for SPU<->CCU communications. This is usually due to main memory not being initialized. Try running **mminit** before the test.

**0xde Failed to open window**
WNDW_OPEN_ERR – An attempt to open the file */dev/wndw* on the SPU disk failed. See if the file exists and verify it has global write permission.

**0xdf Failed to allocate map registers**
IOCTL_ERR – An attempt to set up map registers to main memory for access from the SPU has failed. This should never happen. Please report it.

**0xe0 IOP load module not found**
The file that is to be loaded into the IOP does not exist. The current directory is searched first and then */mnt/test* is searched.

**0xe1 Error attempting to load iop**
The OS utility */mnt/os/loadccu* returned non-zero when it attempted to load the IOP. The usual cause is that *mminit* or a *sysreset* was not performed prior to running the test.

**0xe2 Main memory allocation error**
MMALLOC_ERR – An attempt to allocate main memory has failed. Try running **mminit** before rerunning the test.

# dev4110

# Multibus SMD Disk Formatter, and Interactive Test

## Overview

The *dev4110* test is capable of formatting, verifying the format, and interactively testing up to 12 Storage Module Drives (SMDs) at the same time. It also allows manual input of a Manufacturer's Defect (MD) map before or after a format is performed.

## Prerequisites and Required Equipment

The *dev4110* test is dependent on the existence and correctness of the SPU disk file */mnt/bin/lib/DB_diskfmt*. Refer to the section, "Disk Parameters File, *DB_diskfmt* Description" in this test description for more information about this file.

The test requires the system configuration in one of the next two tables depending on the type of machine under test.

**Table dev4110–1, Hardware Requirements (C1, C120)**

| ITEM | MINIMUM | MAXIMUM |
|---|---|---|
| Xylogics 450/451 controller | 1 | 12 |
| SMD device | 1 | 12 |
| Input/output processor (IOP) | 1 | 5 |
| Multibus card cage | 1 | 10 (2/IOP) |
| Service processor unit (SPU) | 1 | 1 |
| Multibus control unit (MBCU) | 1 | 12 |
| Memory control unit (MCU) | 1 | 1 |
| Memory array unit (MAU) | 4MB | System limit |

**Table dev4110–2, Hardware Requirements (C200 Series)**

| ITEM | MINIMUM | MAXIMUM |
|---|---|---|
| Xylogics 450/451 controller | 1 | 12 |
| SMD device | 1 | 12 |
| Input/output processor (IOP) | 1 | 12 |
| Multibus card cage | 1 | 12 |
| Service processor unit (SP2) | 1 | 1 |
| Multibus control unit (MBCU) | 1 | 12 |
| Memory system (one odd, one even) | 1 | System Limit |
| Peripheral interface adapter (PIA) | 1 | 4 |
| CPU utilities (CPX) | 1 | 1 |

# Test Invocation

The *dev4110* 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 *dev4110* 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 dev4110–1, *dev4110* Test Invocation Sequence**

```
(spu)> cd /mnt/test RETURN
(spu)> sysreset RETURN
(spu)> mminit -s RETURN
(spu)> dshell RETURN
: test dev4110 [-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 dev4110** executes all *dev4110* subtests sequentially except for Class 2 (same as Subtest 200). 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*.

**NOTE**

The Class 2 (Subtest 200) Interactive Test is only started by specifying the **-c** or **-s** options. Only the Class 1 subtest runs if only **test dev4110** is entered.

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 powerup. However, if any problems have occurred subsequent to the last time *initall* was run, (i.e., system crash or hard error), it should be run again. In this case, failure to run *initall* could result in invalid test results. The *initall* utility requires a significant amount of time (2 to 3 minutes depending on if the control stores have been previously loaded) to execute. The control stores do not need to be loaded for disk tests. 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 dev4110–2, Alternate Test Invocation Sequence**

```
(spu)> cd /mnt/test RETURN
(spu)> initall RETURN
(spu)> dshell RETURN
: test dev4110 [-c [class number(s)]] [-s [subtest number(s)]] [+>filename] RETURN
```

## 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 dev4110–3, Getting Help During Test Parameter Entry

| Character | Description |
|---|---|
| ? | Displays this help menu |
| d | Displays *DB_diskfmt* file containing the drive parameters |
| e | Displays entered drive entries |
| h | Provides help for a specific prompt |
| i | Displays the */ioconfig* file |

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

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

## Figure dev4110–3, 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
?       Prints an additional help menu
 1:  Number of errors allowed per device each subtest
    [0-65535]                                        (0) -> RETURN
 2:  Number of devices failed after which test aborts
    [0-65535]                                       (12) -> RETURN
 3:  Verbosity of test [0-65535]                     (3) -> RETURN
 4:  Ioconfig file []                      (/ioconfig) -> RETURN
                PERIPHERAL CONFIGURATION DATA[1]
      CCU    Chassis  Type     CSR    Int Unit Type
    -------- ------- -------- ------ --- ---- --------
 1) iop  4      0    DKC-001  0x3f0  2   0   DKD-008
 2) iop  4      0    DKC-001  0x3f0  2   1   DKD-005
 3) iop  4      0    DKC-001  0x3f8  2   1   DKD-005
 4) iop  4      0    DKC-001  0x3f8  2   0   DKD-008
 5) iop  4      0    DKC-001  0x3f8  2   1   DKD-008
Enter device 99 to begin user-defined configurations or 0 to end selection
 5:  Device selection [0,1-5,99]                     (0) -> 1 RETURN
 6:  Device selection [0,1-5,99]                     (0) -> 99 RETURN
Enter IOP -1 to end user-defined configurations
 7:  IOP [0,3-7]                                     (0) -> 3 RETURN
 8:  Multibus Chassis [0-3]                          (0) -> RETURN
 9:  Controller Offset in Multibus [0x0-0xfff]
                                                (0x3f0) -> 3f8 RETURN
10:  Interrupt number [0-7]                          (2) -> 3 RETURN
11:  Unit number [0-3]                               (0) -> RETURN
12:  Drive name []                            (DKD-005) ->[2] RETURN
Enter IOP -1 to end user-defined configurations
13:  IOP [0,3-7]                                     (0) -> 1 RETURN
14:  Enter OK, or :NN to return to question NN [OK]
                                                    (OK) -> RETURN

          **********  WARNING!  **********
       THIS TEST IS DATA DESTRUCTIVE!
       IF YOU CONTINUE, DATA WILL BE LOST!
************************************************************
* SET WRITE PROTECT ON ALL DRIVES YOU DON'T WANT FORMATTED *
************************************************************
Do you want to continue [yn]  -> y RETURN
```

[1] This information is only displayed once; to display it again, type **i** RETURN at any prompt during test parameter query.

[2] The drive name must match with a drive in the disk parameters file */mnt/bin/lib/DB_diskfmt*). To display the disk parameters file, type **d** RETURN at any prompt.

---

At any time during the test parameter sequence, several options are available as denoted at the

top of the Test Parmeter 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

**?** — Request help for the current prompt (if available)

**^** — Return to the previous prompt

## Prompt Explanations

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

```
Number of errors allowed per device each subtest
[0-65535]                                    (0) ->
```

The number of errors that can occur during a subtest and still allow a device to continue can be specified. For instance, if **3** is entered, a device can have 0-3 errors in any subtest and continue running. On the fourth error, the device stops and is not restarted for the duration of the test. In this example, as long as no more than 3 errors in each subtest occur, the device continues.

> **NOTE**
>
> Subtest 200, Interactive Test reports errors but does not increment the error count when a drive or controller error occurs. In addition, Subtest 100, Input Defects Before Formatting does not affect the error count during pattern test for sector related errors.

```
Number of devices failed after which test aborts
[0-65535]                                   (12) ->
```

Each time a unit fails (exceeds the number of errors allowed each subtest), the count of the number of failed devices is incremented. When a controller fails, the count is incremented for the controller and for each active drive on the controller. If this count exceeds the number of devices specified at this prompt, the entire *dev4110* test aborts. The default number is 12 so the test normally continues even though drives have failed.

> **NOTE**
>
> Drive errors are not counted during the Subtest 200, Interactive Test so failures are not counted during this test.

```
Verbosity of test [0-65535]                 (3) ->
```

The amount of information to be printed can be selected in addition to the standard subtest execution lines. Choose values of the items to print from the table below and add the values. The

sum is the number to enter.

**Table dev4110–4, Test Verbosity Levels**

| VALUE | DESCRIPTION |
|---|---|
| 1 | Print summary of all bad sectors found – printed near the end of Subtest 101, Format and Pattern Test. |
| 2 | Print each bad sector as it is found in Subtest 101 – may print same sector once for each pattern. |
| 4 | Print read of the BBT at startup of Subtest 103, Verify System Format, Subtest 104 Test Diagnostic Cylinder, and Subtest 200, Interactive Test. |
| 8 | Print pattern and pass counter in Subtest 200, Interactive Test pattern test. Only the pass counter is normally displayed as an indicator of how far along the test is toward completion. |

**NOTE**

If each bad sector is to be printed as it is found, the following errors are printed.

- 0x05 (header not found)
- 0x06 (Hard ECC error)
- 0x1e (Correctable ECC error)

Any other reported errors than those listed above are countable errors and increment the number of errors per subtest counter. Countable errors are always retried until successful or the number of errors per subtest counter exceeds the number allowed.

The following is an example of verbose output for a value of 1:

**Figure dev4110–4, Verbose Output Screen — Level 1**

```
SUMMARY OF BAD BLOCKS FOR ccu/mb/csr/d=3/0/3f0/0

 INPUTS THAT WILL NOT BE SLIPPED:
 CYL HD SEC   BCAI   LEN LOC  ERR SRC | CYL HD SEC   BCAI   LEN LOC  ERR SRC
         ---------------------- no sectors ----------------------

LOC:  *G=gap  *D=dup  *S=spare  *I=invalid hdr  *B=hdr marked bad

 INPUTS THAT WILL BE SLIPPED:
 CYL HD SEC   BCAI   LEN LOC  ERR SRC | CYL HD SEC   BCAI   LEN LOC  ERR SRC
         ---------------------- no sectors ----------------------

LOC:  *G=gap  *D=dup  *S=spare  *I=invalid hdr  *B=hdr marked bad

SUMMARY OF BAD BLOCKS FOR ccu/mb/csr/d=3/0/3f0/1

 INPUTS THAT WILL NOT BE SLIPPED:
 CYL HD SEC   BCAI   LEN LOC  ERR SRC | CYL HD SEC   BCAI   LEN LOC  ERR SRC
 ----------------------------- no sectors ---------------------------------

LOC:  *G=gap  *D=dup  *S=spare  *I=invalid hdr  *B=hdr marked bad

 INPUTS THAT WILL BE SLIPPED:
 CYL HD SEC   BCAI   LEN LOC  ERR SRC | CYL HD SEC   BCAI   LEN LOC  ERR SRC
  22  3   7                  SECC PGM | 372 12   0                  HECC PGM
  51 14  28                  HECC PGM | 523 18   2                  SECC PGM
  77  9  44                  SECC PGM | 544  6  11                   HNF PGM
  78  9  44                   HNF PGM | 591  0   7                  HECC PGM
  79  9  44                  SECC PGM |

LOC:  *G=gap  *D=dup  *S=spare  *I=invalid hdr  *B=hdr marked bad
```

The following is an example of verbose output for a value of 2:

## Figure dev4110–5, Verbose Output Screen — Level 2

```
    subtest 101  0:22:14  Pattern 1: 00000000

dev4110 ERROR (0x1E)-ccu/mb/csr/d=3/0/3F0/1 Correctable ECC error
  Cyl:  22 Hd: 3 Sect:  7
 0:23:52

dev4110 ERROR (0x06)-ccu/mb/csr/d=3/0/3F0/1 Hard ECC error
  Cyl:  51 Hd:14 Sect: 28
 0:29:45 Pattern 2: FFFFFFFF

  ----- More errors will probably be reported during other patterns.
  ----- In fact the errors shown above will probably repeat.
```

The following is an example of verbose output for a value of 4:

## Figure dev4110–6, Verbose Output Screen — Level 4

```
--------------- BAD BLOCK TABLE READS -----------------
Drive: ccu/mb/csr/d - 3/0/3f0/0
        Reading copy #0 starting at cyl:758 hd:18 sec:34    - failed
        Reading copy #1 starting at cyl:758 hd:18 sec:39    - successful
Drive: ccu/mb/csr/d - 3/0/3f0/1
        Reading copy #0 starting at cyl:758 hd:18 sec:34    - successful
```

The following is an example of verbose output for a value of 8:

## Figure dev4110–7, Verbose Output Screen — Level 8

```
(D1;Save)-> p 100 times 0 0 0 to end
pass: 1 pattern: ffffffff
```

With a verbosity setting of 8, the `pass: 1` and the `pattern: ffffffff` line print. Both the pass number and pattern change as the test progresses.

```
Ioconfig file []                    (/ioconfig) ->
```

This prompt allows a file to be chosen that contains peripheral device descriptions. The default displays the system's configuration file. To display an I/O configuration file previously created, enter the filename. Then the configuration file is displayed. (A relative or absolute path may be specified.)

If drives are to be formatted or tested with a configuration other than the expected configuration, a new I/O configuration file may be created. As an alternative, nonstandard configurations may be input by responding to the user-defined configuration test parameters (questions 7-13 in the "Test Parameter Menu" figure). Then the drive and/or controller does not have to be defined in an I/O configuration file. Drives can be selected from an I/O configuration file and then drives can also be defined with user-defined configurations.

```
Device selection [0,1-5,99]                (0) ->
```

Enter the number of the drive to be tested. One drive can be selected with each prompt for device selection and up to a total of 12 drives can be selected in any order. Refer to the *Hardware Requirements (C1, C120)* table and the *Hardware Requirements (C200 Series)* table for restrictions on hardware configurations.

> **NOTE**
>
> If drives are formatted for system use, the drives should be on separate controllers. Each controller must format drives sequentially so two on one controller doubles the format time.

If **0** is entered, the selection is terminated and the prompt `Enter OK` is displayed. If **99** is entered, the user-defined prompts are displayed, beginning with the following prompt.

```
IOP [0,3-7]                                (0) ->
```

This prompt begins the user-defined drive prompts, so if all the drives to be used are in the *ioconfig* file, reply RETURN to end the user-defined drive prompts. However, to enter a unique drive configuration (not in the *ioconfig* file), enter the CCU slot number for the desired IOP. Then additional prompts are displayed requesting hardware configuration information.

```
Multibus Chassis [0-3]                     (0) ->
```

Enter the number of the chassis to be tested. (The drive is attached to a controller which is in a Multibus chassis.)

```
Controller Offset in Multibus [0x0-0xfff]
                                          (0x3f0) ->
```

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

```
Interrupt number [0-7]                    (2) ->
```

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

```
Unit number [0-3]                         (0) ->
```

Enter the unit number of the drive to be tested. The unit number to enter is also the unit selected on the drive (Up to four drives can be attached to one controller; each drive is assigned a unit number, which is usually switch selectable on the drive). For maximum performance, each drive should be on a separate controller although it is not a requirement.

```
Drive name                          (DKD-005) ->
```

Enter the drive name. The response for this query must be in the form of DKD-xxx and the name must match one of the drive definitions found in the disk parameters file (*/mnt/bin/lib/DB_diskfmt*).

```
IOP [0,3-7]                               (0) ->
```

Enter the CCU slot number for the desired IOP to set up test parameters for another user-defined drive. Enter a **-1** or RETURN to terminate test parameter entry.

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

To return to a preceding question at anytime during the test parameter sequence, enter CTRL and RETURN. To return to a specific question and make any further changes, a colon and a question number may be entered, for example, **:3**.

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

```
Do you want to continue [yn]          ->
```

Enter RETURN to begin test execution; or enter **n** to abort the *dev4110* test.

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 dev4110–8, Sample Test Parameter Summary**

```
            TEST PARAMETER SUMMARY

Number of errors allowed per device each subtest      : 0
Number of devices failed after which test aborts      : 12
Verbosity of test                                     : 3
Ioconfig file                                         : /ioconfig
Enter OK, or :NN to return to question NN             : OK

                    DRIVE CONFIGURATION DATA

      IOP MBus MBus  Int   Unit  #    #  Phys  Log
       #   #   CSR  Level   #   Cyl  Hds  Sec  Sec  Name
      --- --- ----- -----  ---  ---  ---  ---  ---  -------
   1.  4   0  0x3f0   2     0  1024   27   68   67  DKD-008
1000.  4   0  0x3f8   3     0   760   19   60   59  DKD-005

Performing sysreset of ccus and memory...
Initializing I/O subsystem and Loading IOP(S)...

************************************
* SYSTEM FORMAT IS ABOUT TO START    *
* VERIFY WRITE-PROTECT IS SET ON ANY *
* DRIVES YOU ARE NOT FORMATTING      *
************************************

Then reconfirm that you want to continue    [yn] ->
```

# 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.

# Class Descriptions

The *dev4110* test contains the following two classes of tests.

**Table dev4110–5, *dev4110* Test Classes**

| CLASS | DESCRIPTION |
|---|---|
| 1 | System format and verification tests |
| 2 | Interactive test |

## Class 1 Subtests

Class 1 subtests allow a system format to be written to an SMD and to be verified. The proper way to format a disk is to execute all subtests in the default order.

> **NOTE**
>
> Some individual functions performed during formatting are contained in two separate subtests in order to assist in special testing of drives. Therefore, Subtest 101, Format and Pattern Test should not be invoked unless Subtest 102, Initialize Diagnostic Cylinder is also invoked.

If only Subtest 101, Format and Pattern Test is invoked, the option to quit or continue is available. The test displays the warning message shown in the following figure if only Subtest 101 is invoked:

**Figure dev4110–9, System Format and Verification Warning Screen**

```
WARNING! SMD Disk Formatter, dev4110, should never be invoked with
         Subtest 101 selected and no Subtest 102.
Do you want to continue [yn]     - y (RETURN)


                                WARNING!!

        YOU HAVE CHOSEN TO ONLY PARTIALLY FORMAT A DRIVE!
        THE DRIVE WILL NOT BE ACCEPTABLE FOR SYSTEM USE!
```

The Class 1 subtests are shown in the following table:

## Table dev4110–6, Class 1 Subtests

| SUBTEST | DESCRIPTION | TIME (hr:min:sec) |
|---|---|---|
| 100 | Input Defects Before Formatting | N/A |
| 101 | System Format of SMD | 2:20:00[2] |
| 102 | Initialize Diagnostic Cylinder | 3:00[1] |
| 103 | Verify System Format of SMD | 5:30[2] |
| 104 | Test Diagnostic Cylinder | :15[1] |

[1] This test is run sequentially on each drive so multiply this time by the number of drives for an estimate of the required time for this subtest.

[2] Times will double, triple, or quadruple for 2, 3 or 4 drives, respectively, per controller.

### Subtest 100, Input Defects Before Formatting

Subtest 100 is used to enter defect map information before pattern testing. The inputs can be entered or can be read from a specified file. In either case, the inputs can be in Byte-Count-After-Index (BCAI) format or in logical sector number format.

A command processor is invoked when this subtest begins. Several commands are available for the input of defects. Startup of this subtest is shown in the following figure:

## Figure dev4110–10, Subtest 100 Startup Screen

```
Type 'h' for help with commands
       Logical sector input mode selected.

If you do not have defects to enter, just enter 'q'

make_bad (D1;Sector)->
```

Enter **h** to display a help facility which lists the available commands. The help output is displayed as shown in the following figure:

## Figure dev4110–11, Subtest 100 Help Screen

```
Commands:
c[hange_mode]        - toggles logical sector/defect map entry mode
cyl hd sec           - logical sector to be slipped
cyl hd bcai len      - defect map entry to be slipped
del[ete] cyl hd sec - delete logical sector entry
del[ete] cyl hd bcai len - delete defect map entry
dr[ive] [n]          - change/display current drive
f[ile] filename      - get inputs from specified file
h[elp]               - display this information
l[ist]               - list inputs made so far
q[uit]               - exit
!UNIX-command        - execute UNIX command
'comment             - comment; ignored
```

When manually entering defects, first select the entry mode. The default mode is logical sector mode. To enter manufacturer's defects (which are in BCAI format), switch modes with the **change_mode** command. Then enter the position of each flaw. In sector mode, enter the cylinder, head, and sector where the flaw exists. In BCAI mode, enter the cylinder, head, byte-count-after-index, and bit-length.

To enter the defect data from one or more files, select each file by typing

**file** *filename*

where *filename* is replaced with the actual name of the file. The first line of the file defines the drive type and looks like the following:

n *drivename*

where *drivename* is the drivename which can be found in the file */mnt/bin/lib/DB_diskfmt* on the Service Processor (for example, DKD-005 for the 1/2 GByte NEC 2352 drive). Following the drive name line, the file has remaining lines that contain either BCAI inputs or logical sector inputs. A BCAI input is preceded with the letter **d** while a logical sector input is preceded with the letter **s**. A spare sector can be specified by using a logical sector number one greater than the last used sector number. To specify a track that is to be mapped to an alternate track, use the letter **m** followed by the cylinder and head of the bad track. One or more spaces separate each item in the file and a new line can be started between any two items. Comments may be included in the file by typing a # at any place on a line. All characters from the # to the end of the line are ignored. Blank lines are also allowed. A file containing defect data is shown in the following figure:

## Figure dev4110–12, Creating a Defect Data File

```
#------------------------------ start of file ----------------------------
# serial number: 003013
n  DKD-005  # NEC 2352 drive
#   cyl  hd   bcai  len       cyl  hd   bcai  len
d   157  15  24395   34    d  297  14  14102    4  # BCAI Inputs
d   466   0    206    1

#   cyl  hd  sec             cyl  hd  sec
s   457  12   42          s  821   5   11         # Logical sector inputs
#------------------------------ end of file -----------------------------
```

**NOTE**

To abort *dev4110* while in this subtest, press CTRL and b. The SPU takes about 30 seconds to dump memory. Then the SPU prompt is displayed. If CTRL and c are pressed, it is intercepted by the subtest and the subtest prompt is displayed.

The current input mode (logical sector or defect map mode) is revealed in the slip prompt. If in logical sector input mode, the prompt appears as follows:

```
make_bad (D1;Sector)->
```

If in defect map input mode, the prompt appears as follows:

```
make_bad (D1;Defect)->
```

When a defect is entered, a list of logical sectors that correspond to the defect is generated as follows:

## Figure dev4110–13, List of Logical Sectors For a Defect

```
make_bad (D1;Defect)-> 203 1 572 9510
        Associated logical sectors: 59 0 1
```

The previous figure illustrates that three sectors are affected by the media flaw which starts at byte 572 from index and is 9510 bits long.

The byte before and after any flaw is also considered bad to avoid the possibility that a flaw might be slightly bigger than is reported in the defect map.

If two defects affect the same sector, the logical sector is only entered in the *INPUTS THAT WILL BE SLIPPED* input list once. Subsequent entries of the same sector are entered in the *INPUTS THAT WILL NOT BE SLIPPED* list.

For instance, if the following two defects exist:

| cylinder | head | bcai | bit-length |
|---|---|---|---|
| 10 | 0 | 70 | 1 |
| 10 | 0 | 80 | 1 |

Both defects fall in sector 0 so when the second defect is entered, it does not create a new entry in the input list. This is shown in the following figure:

**Figure dev4110–14, Two Defects Entered for the Same Sector**

```
make_bad (D1;Defect)-> 10 0 70 1
     Associated logical sectors: 0
make_bad (D1;Defect)-> 10 0 80 1
     Associated logical sectors: (0)
```

The following is an example of entering a bad location and deleting it:

**Figure dev4110–15, Entering and Correcting An Incorrect Location**

```
make_bad (D1;Sector)-> 17 7 23
make_bad (D1;Sector)-> 17 7 46
make_bad (D1;Sector)-> d 17 7 46
        Input 17 7 46 has been removed
make_bad (D1;Sector)-> list

 INPUTS THAT WILL NOT BE SLIPPED:
 CYL HD SEC   BCAI   LEN LOC  ERR SRC | CYL HD SEC   BCAI   LEN LOC  ERR SRC
         ---------------------- no sectors ----------------------

 INPUTS THAT WILL BE SLIPPED:
 CYL HD SEC   BCAI   LEN LOC  ERR SRC | CYL HD SEC   BCAI   LEN LOC  ERR SRC
 17  7  23
```

The following is an example of switching drives and entering defects for the drive from a file:

## Figure dev4110–16, Entering Defects for a New Disk Drive

```
make_bad (D1;Sector)-> dr 2
        Drive 2 is now selected
make_bad (D2;Sector)-> !cat SN12345678          <-to illustrate unix cmd
# Serial Number: 12345678
d  100  5 10536     1
d  257  9 26401  1000
> file SN12345678                                <-prompt after unix cmd is '>'
make_bad (D2;Sector)-> list

 INPUTS THAT WILL NOT BE SLIPPED:
 CYL HD SEC   BCAI   LEN LOC  ERR SRC | CYL HD SEC   BCAI   LEN LOC  ERR SRC
         ---------------------- no sectors ----------------------

 INPUTS THAT WILL BE SLIPPED:
 CYL HD SEC   BCAI   LEN LOC  ERR SRC | CYL HD SEC   BCAI   LEN LOC  ERR SRC
 100  5  11  10536     1      N/A USR | 257  9  35  26401  1000      N/A USR
 257  9  36  26401  1000      N/A USR |

make_bad (D2;Sector)->
```

In the above example, the manual mode is set to `Sector` but the file contains BCAI information. This is does not create a problem. The manual mode only applies to manual inputs. File inputs are controlled by the character that precedes the defect location. The defect at cylinder 257, head 9 was large enough to overlap two sectors so both were flagged bad.

Type **quit** after entering all inputs to exit the subtest. Then Subtest 101 starts.

### Subtest 101, System Format of SMD

Subtest 101 formats a drive for system use except for the creation of the diagnostic cylinder which is performed in Subtest 102, Initialize Diagnostic Cylinder. This subtest consists of the following five parts:

1. Format with no spare sectors to allow pattern test of all sectors.
2. Pattern test by first reading the `0x00000000` pattern which was written during format. Then write and read seven other patterns which are shown below:

   `0xAAAAAAAA` `0xAF5BAF5B`
   `0xFFFFFFFF` `0x5EB75EB7`
   `0xEBD6EBD6` `0x6DB66DB6`
   `0xD7ADD7AD`

   These patterns can be replaced by up to 16 other patterns if the file *dev4110.patt* is created in the current directory or if it is created in */mnt/bin/lib*. The current directory is searched first. The format of the file is free form with eight hex characters per pattern and at least one space separating the patterns. More than one line can be used to define the patterns. Blank lines are also allowed.

Any sector-related errors (header not found; hard ECC error; or correctable ECC error) are retried immediately and, if an error occurs again in ten retries (does not have to be the same error), the sector location is saved for later slipping. A maximum of 1365 errors can be saved per drive. The maximum includes the flaws from Subtest 100, Input Defects Before Formatting. Errors that do not repeat are discarded. If a verbose value of 2 is selected, each error is printed at the time it is detected.

3. Format with one spare sector per track.
4. Slip sectors which were input in Subtest 100, Input Defects Before Formatting or which were detected as bad during pattern test. If more than one sector is bad on a track, one sector is slipped and then the remaining bad sectors are added to the bad block table. This table associates the bad sectors with good sectors located in the last and possibly next to the last cylinder. Up to 639 sectors can be relocated.

   If a verbose value of 1 (default) is selected in the test parameter menu, a summary of bad sectors for each drive is printed before slipping occurs.
5. The last operation is to write five copies of the bad block table to the last 25 sectors of the disk.

### Subtest 102, Initialize Diagnostic Cylinder

Subtest 102 writes special patterns to the diagnostic cylinder which is the cylinder just before the cylinder used for relocated sectors. Sector 0 of each track contains a table of contents that describes what is written to each of the remaining sectors. If no sectors have been relocated on a track, sector 1 is marked bad. The next twelve sectors contain Error Correction Code (ECC) errors of length 1 to 12 respectively. The remaining sectors have the seven patterns listed in Subtest 101, System Format of SMD written to them. Sector 13 (or 14 if sector 1 is marked bad) will contain an `0xAAAAAAAA` pattern, sector 14 (or 15) will contain an `0xFFFFFFFF` pattern, and so on with the patterns repeating every seven sectors. Refer to the section Disk Parameters File, *DB_diskfmt* Description for more information about the diagnostic cylinder.

### Subtest 103, Verify System Format of SMD

Subtest 103 reads every logical sector in the usable portion of the disk. The usable portion of each drive is all the cylinders preceding the diagnostic cylinder and all relocated sectors (which are found on the innermost one or two cylinders). When a header not found error occurs, the bad block table is checked to see if the header has been relocated. If it has been relocated, the alternate sector is read.

Any errors other than those due to sector relocation are reported and count against the device errors per subtest limit. If the number of errors exceeds the limit, the drive is failed.

### Subtest 104, Test Diagnostic Cylinder

Subtest 104 reads each sector on the diagnostic cylinder and expects soft or hard ECC errors on sectors 1 through 12 of each track (or sectors 2 through 13 if sector 1 is marked bad). The remaining sectors must be error-free.

## Class 2 Subtest

The Class 2 subtest performs interactive testing of all selected drives. The Class 2 subtest can only be started by explicitly using the **-c** or **-s** options when *dev4110* is invoked. The Class 2 subtest is shown in the following table.

**Table dev4110–7, Class 2 Subtest**

| SUBTEST | DESCRIPTION | TIME (min:sec) |
|---|---|---|
| 200 | Interactive Test | N/A |

### Subtest 200, Interactive Test

Subtest 200 provides a special interactive test interface to perform the following disk maintenance tasks:

- Pattern test portions of the disk
- Slip bad sectors
- Format single tracks
- Several more useful operations

A help facility is available when needed.

> **CAUTION**
>
> Three of the interactive test commands are potentially data destructive: **format**, **pattern_test**, and **slip_sectors**.

Each command performs an automatic save of the data it is to overwrite before the operation begins. However, the commands can still be data destructive if one of the following actions is attempted:

1. An attempt to pattern test more than one track is performed and **y** is entered when asked to confirm multiple track pattern test (because only one track can be saved).
2. An attempt to preserve a track's data before a destructive operation fails and a response is entered to force the operation anyway.
3. The pattern test, slip sectors, or format operation has completed and then data errors occur during the restore of the track's data.
4. The keys CTRL and **c** or CTRL **b** are pressed in the middle of the operation.
5. Automatic save is disabled by using the **toggle_save** command.
6. A power outage or some other unforeseen event causes the test to abort in the middle of the operation.

### Interactive Test Invocation

To invoke the interactive test, use the procedure shown in the following figure:

**Figure dev4110–17, Interactive Test Invocation Sequence**

```
(spu)> cd /mnt/test
(spu)> mminit -s
(spu)> dshell
: test dev4110 -c 2 (or test dev4110 -s 200)
```

### Interactive Test Parameter Menu

The prompts are displayed as shown in the following figure. Use defaults for the first four prompts and then select the drives to test and press RETURN after each drive is selected. After all drives have been selected, press RETURN twice and enter **y**. After about a minute, the interactive test will start.

The following figure shows the prompts for the Interactive Test Parameter Menu:

## Figure dev4110–18, Interactive 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

          ?       Prints an additional help menu
 1:  Number of errors allowed per device each subtest
    [0-65535]                                          (0) -> RETURN
 2:  Number of devices failed before test aborts
    [0-65535]                                         (12) -> RETURN
 3:  Verbosity of test [0-65535]                       (3) -> RETURN
 4:  Ioconfig file                            (/ioconfig) -> RETURN

                PERIPHERAL CONFIGURATION DATA[1]
     IOP  MBUS  TYPE       CSR    INT  UNIT  TYPE
     ---  ----  --------  -----   ---  ----  --------
 1)   3     0   DKC-001   0xdc0    1     0   DKD-001
 2)   3     0   DKC-001   0xdc8    4     0   DKD-001
 3)   6     1   DKC-001   0x3f0    3     0   DKD-001

Enter device 99 to begin user-defined configurations or 0 to end selection

 5:  Device selection [0,1-3,99]                       (0) -> 1 RETURN
 6:  Device selection [0,1-3,99]                       (0) -> 2 RETURN
 7:  Device selection [0,1-3,99]                       (0) -> 3 RETURN
 8:  Device selection [0,1-3,99]                       (0) -> RETURN
 9:  Enter OK, or :NN to return to question NN [OK]
                                                      (OK) -> RETURN

   **********  WARNING!  **********
THIS TEST IS POTENTIALLY DISK DATA DESTRUCTIVE!
IF YOU CONTINUE, DATA COULD BE LOST!
Do you want to continue [yn]      -> y RETURN
```

[1] This information is only displayed once; to display it again , type i RETURN at any prompt during test parameter query.

### Interactive Test Mode

After the test parameter menu prompts have been answered the following is displayed indicating interactive test mode:

> **NOTE**
>
> In the following figure, D1 indicates drive 1 is selected and Save indicates that track data is saved during the data destructive commands **format**, **pattern_test**, and **slip_sectors**.

## Figure dev4110–19, Interactive Test Mode

```
                    INTERACTIVE TEST MODE
                      FOR SMD DRIVES

Type 'h' for help
(D1;Save)->
```

Type **h** to list the available interactive test commands. For additional help, refer to the command descriptions in the following section. Only a unique portion of the beginning of a command needs to be typed.

To exit the interactive test, type **q** and press RETURN. If a track is saved, a prompt is displayed indicating whether data can be lost. If **y** is entered or if a track is not saved, *dev4110* ends normally and exits back to the dshell prompt. If **n** is entered, the test returns to the interactive prompt without executing changes.

### Interactive Test Commands

The following table lists the available commands and also a description of each command.

**Table dev4110–8, Interactive Test Commands**

| COMMAND | DESCRIPTION |
|---|---|
| **a[lternate_seek]** | Seek between two selected cylinders |
| **b[bt_display]** | Display Bad Block Table |
| **d[rive_select]** | Select a drive |
| **f[ormat]** | Reformat one track |
| **hd[e_display]** | Display header, data, and ECC |
| **h[elp]** | Display list of commands and their arguments |
| **i[nitialize_bbt]** | Rebuild damaged Bad Block Table |
| **p[attern_test]** | Pattern test range of sectors |
| **q[uit]** | Exit from the Interactive Test |
| **r[estore_track_data]** | Restore previously saved track data |
| **sa[ve_track_data]** | Save one track's data |
| **sc[an_flaws]** | Produce file of all marked-bad sectors |
| **se[ctor_display]** | Display data from range of sectors |
| **sl[ip_sectors]** | Slip one or more sectors |
| **st[atus]** | Display current drive and which track is saved |
| **to[ggle_save]** | Switches between allowing and disabling auto-save |
| **tr[ack_headers_display]** | Display track headers |
| **v[erify_format]** | Read a range of sectors |
| **!UNIX-command** | Execute UNIX command |
| **'comment** | Any comment; ignored |

## Interactive Test Commands Descriptions

The following sections describe the interactive test commands.

### Seek Between Two Selected Cylinders

**a[lternate_seek]**

This command performs seeks between two selected cylinders.

### Display Bad Block Table

**b[bt_display]**

This command displays the active drive's bad block table as follows:

## Figure dev4110–20, Active Drive's Bad Block Table Display Screen

```
BAD BLOCK TABLE FOR ccu/mb/csr/d=3/0/3f0/0

        CYL HD SEC  CYL HD SEC  CYL HD SEC  CYL HD SEC  CYL HD SEC
        275  3  48  387 11  48
```

When one bad sector is slipped on a track, the bad sector is slipped and all subsequent sector headers are shifted by one with the last logical sector replacing the spare sector. Additional slipped sectors on a track causes the associated headers to be marked bad and the sectors to be relocated. To display which sectors are bad, use the **track_headers_display** command. The bad sectors are flagged with a *B. To display a different drive's bad block table, use the **drive_select** command to change drives.

### Select a Drive

**d[rive_select]** [*drive-number*]

This command can change which drive is the active drive. Most of the other commands operate on the active drive only. The argument *drive-number* is the entry number from the Drive Configuration Data which was displayed at the end of the user prompts. To display the entry numbers, enter **d** and press RETURN. The Drive Configuration Data and active drive are then redisplayed, and a prompt is displayed to enter a new drive entry number. Enter **0** or press RETURN for the current drive.

Use of the **drive_select** command is shown in the following figure:

**Figure dev4110–21, Drive Configuration Data Screen**

```
(D1;save)-> d

                    DRIVE CONFIGURATION DATA

    IOP MBus MBus  Int   Unit  #    #   Phys  Log
     #   #   CSR  Level   #   Cyl  Hds  Sec   Sec  Name
    --- --- ----- -----  ---  ---  ---  ---   ---  -------
 1.  3   0  0x3f0   2     0   760   19   60    59  DKD-005
 2.  3   0  0x3f0   2     1   760   19   60    59  DKD-005
 3.  3   0  0x3f8   3     0   842   20   46    45  DKD-001

       Drive 1 is currently selected
       Enter new selection or 0 to leave it as is (0,1-3) [0] -> 2
       Drive 2 is now selected
(D2;save)-> d 3
       Drive 3 is now selected
(D3;save)->
```

The configuration data does not display if the drive number is entered. Also, the drive number is redisplayed with each prompt as D$x$ where $x$ is the current drive number.

### Reformat One Track

**f[ormat]** *cyl hd*

This command unslips sectors or recreates a bad track's headers. If automatic save is enabled (the prompt indicates Save), then before the track is reformatted, an attempt is made to read every logical sector (including relocated sectors). If a read fails, the options of retrying the sector, terminating the format before any destructive operation occurs, or forcing the format even though one or more sectors may not be recovered are displayed.

> **CAUTION**
>
> If auto save is disabled (Nosave mode) by using the **toggle_save** command, the track is reformatted without preserving the track's data.

In the following figure, cylinder 300, head 12 is reformatted on an NEC drive (760 cylinders, 19 heads, 59 logical sectors). The track contains two slipped sectors as shown. The second bad sector has been relocated.

**CAUTION**

If auto save is enabled and sectors are unslipped by the **format** command, data is restored to the bad sectors which could cause the data to no longer be readable. This is okay since the data is still saved. Slip the bad sectors using the **slip_sectors** command and then use the **restore_track_data** command to restore the track's data.

The following figure shows how to perform the restoration operation:

**Figure dev4110–22, Formatting a Track**

```
(D3;Save)-> format 300 12
         No data has been previously saved. Saving this track
         Data save operation complete

  TRACK HEADERS BEFORE FORMAT:
 cylinder = 300  head = 12
 47 48 49 50 51 52 53 54 55 56 57 58  0  1  2  3 *B  4  5  6  7  8  9 10 11
 12 13 14 *B 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36
 37 38 39 40 41 42 43 44 45 46

         About to format cylinder 300  head 12. Are you sure? [yn] -> y
         Format complete.  2 sectors have been unslipped.
         Restoring data to track
         Data restore operation complete
(D3;Save)-> track 300 12
 cylinder = 300  head = 12
 48 49 50 51 52 53 54 55 56 57 58 *S  0  1  2  3  4  5  6  7  8  9 10 11 12
 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37
 38 39 40 41 42 43 44 45 46 47
(D3;Save)->
```

Then pattern test the track and identify which sectors are bad, and slip the bad sectors. Then restore the original track data. These operations are shown in the following figure:

## Figure dev4110–23, Restoring the Original Track Data

```
(D3;Save)-> patt 10 times 300 12 0 to 300 12 58
     Patterns used:
      aaaaaaaa ffffffff ebd6ebd6 d7add7ad af5baf5b 5eb75eb7 6db66db6
        If you save this track, you will lose the saved data for
        drive 3: cylinder 300  head 12.
        Save data? [yn] -> n                <- previous data is what you want
        WARNING - DATA NOT SAVED

 pass # 1
dev4110 ERROR (0x1e) ccu/mb/csr/d 3/0/3f0/0 - Correctable ECC error
 cyl: 300 hd: 12 sec: 4

dev4110 ERROR (0x06) ccu/mb/csr/d 3/0/3f0/0 - Hard ECC error
 cyl: 300 hd: 12 sec: 16

---- more errors will be detected as the test loops. ----

(D3;Save)-> slip
slip (D3;Save;Sector)-> 300 12 4
slip (D3;Save;Sector)-> 300 12 16
slip (D3;Save;Sector)-> e

 INPUTS THAT WILL NOT BE SLIPPED:
 CYL HD SEC   BCAI   LEN LOC  ERR SRC | CYL HD SEC   BCAI   LEN LOC  ERR SRC
         ---------------------- no sectors ----------------------

 INPUTS THAT WILL BE SLIPPED:
 CYL HD SEC   BCAI   LEN LOC  ERR SRC | CYL HD SEC   BCAI   LEN LOC  ERR SRC
 300 12   4                   N/A USR | 300 12  16                   N/A USR

        Are you sure inputs are ready for slipping? [yn] -> y
      sector 300 12  4 and
             300 12 16: slipped
(D3;Save)-> restore
        Data has been saved for drive 3: cylinder 300 head 12
        Do you really want to restore this data? [yn] -> y
        Data restore operation complete
(D3;Save)->
```

### Display Header, Data, and ECC

**hd[e_display]** *cyl hd sec* [**to** *cyl hd sec*]

This command is used to display the sector data, the header, and ECC. The data is read in raw mode; therefore, the controller does not perform error checking.

> **NOTE**
>
> The sector specified for the **hde_display** command is the physical sector numbered from index. All other commands search all headers on the track until the requested header is found.

To illustrate the **hde_display** command, the track headers displayed by the **track** command and the sector displayed by the **hde_display** command are shown in the following figure:

**Figure dev4110–24, *hde_display* Command**

```
(D2;Save)-> track 27 5
 cylinder =  27  head =  5
 55 56 57 58 *S  0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19
 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44
 45 46 47 48 49 50 51 52 53 54
(D2;Save)-> hde 27 5 0

Phys:  27  5   0  Log:  27  5  55  Type: 0        Header:001b0537 ECC:2bc18e26
000  6db66db6 6db66db6 6db66db6 6db66db6  6db66db6 6db66db6 6db66db6 6db66db6
020  6db66db6 6db66db6 6db66db6 6db66db6  6db66db6 6db66db6 6db66db6 6db66db6
040  6db66db6 6db66db6 6db66db6 6db66db6  6db66db6 6db66db6 6db66db6 6db66db6
060  6db66db6 6db66db6 6db66db6 6db66db6  6db66db6 6db66db6 6db66db6 6db66db6
080  6db66db6 6db66db6 6db66db6 6db66db6  6db66db6 6db66db6 6db66db6 6db66db6
0a0  6db66db6 6db66db6 6db66db6 6db66db6  6db66db6 6db66db6 6db66db6 6db66db6
0c0  6db66db6 6db66db6 6db66db6 6db66db6  6db66db6 6db66db6 6db66db6 6db66db6
0e0  6db66db6 6db66db6 6db66db6 6db66db6  6db66db6 6db66db6 6db66db6 6db66db6
100  6db66db6 6db66db6 6db66db6 6db66db6  6db66db6 6db66db6 6db66db6 6db66db6
120  6db66db6 6db66db6 6db66db6 6db66db6  6db66db6 6db66db6 6db66db6 6db66db6
140  6db66db6 6db66db6 6db66db6 6db66db6  6db66db6 6db66db6 6db66db6 6db66db6
160  6db66db6 6db66db6 6db66db6 6db66db6  6db66db6 6db66db6 6db66db6 6db66db6
180  6db66db6 6db66db6 6db66db6 6db66db6  6db66db6 6db66db6 6db66db6 6db66db6
1a0  6db66db6 6db66db6 6db66db6 6db66db6  6db66db6 6db66db6 6db66db6 6db66db6
1c0  6db66db6 6db66db6 6db66db6 6db66db6  6db66db6 6db66db6 6db66db6 6db66db6
1e0  6db66db6 6db66db6 6db66db6 6db66db6  6db66db6 6db66db6 6db66db6 6db66db6

(D2;Save)->
```

The previous figure shows that the physical sector 0 corresponds to logical sector 55 on this track.

### Display List of Commands and Arguments

**h[elp]**

This command lists the commands and their arguments.

### Rebuild Damaged Bad Block Table

**i[nitialize_bbt]**

This command reads the Bad Block Table (BBT) and attempts the following:

1. Tries to read (successfully)any one of the five copies in the BBT
2. If any one of the copies is read successfully, it is used to recreate all of the other copies
3. If none of the copies in the BBT are read successfully, a message is printed out stating that all copies of the BBT are bad. The following prompt is then printed:

```
Do you want to retry the last operation [yn] ->
```

If this prompt is answered with **y**, program will attempt to read (successfully) the BBT again. If **n** if entered, the following prompt will be displayed:

```
WARNING:  This could result in several NEW Header Not Found Errors.

Do you want to continue [yn] ->
```

> **WARNING**
>
> If the previous prompt is answered with **y**, any previous entries to the BBT will be erased. Run *verify_format* and take appropriate action if any Header Not Found messages appear.

The following figure displays an example of this command:

## Figure dev4110–25, Initialize_bbt Example

```
(D1;Save)-> initialize_bbt
Drive 1: BBT copy 1 is bad.  Count field is 2779096485. Should be 0-638.
Drive 1: BBT copy 2 is bad.  Count field is 2779096485. Should be 0-638.
Drive 1: BBT copy 3 is bad.  Count field is 2779096485. Should be 0-638.
Drive 1: BBT copy 4 is bad.  Count field is 2779096485. Should be 0-638.
Drive 1: BBT copy 5 is bad.  Count field is 2779096485. Should be 0-638.


dev4110 ERROR (0x4d)-ccu/mb/csr/d=7/3f0/0 All copies of BBT are bad
The internal copy of the BBT is cleared since no good copies were found
Do you want to retry the last operation [yn] -> y
Drive 1: BBT copy 1 is bad.  Count field is 2779096485. Should be 0-638.
Drive 1: BBT copy 2 is bad.  Count field is 2779096485. Should be 0-638.
Drive 1: BBT copy 3 is bad.  Count field is 2779096485. Should be 0-638.
Drive 1: BBT copy 4 is bad.  Count field is 2779096485. Should be 0-638.
Drive 1: BBT copy 5 is bad.  Count field is 2779096485. Should be 0-638.


dev4110 ERROR (0x4d)-ccu/mb/csr/d=7/3f0/0 All copies of BBT are bad
The internal copy of the BBT is cleared since no good copies were found
Do you want to retry the last operation [yn] -> n

WARNING: This could result in several NEW Header Not Found Errors.

Do you want to continue [yn] -> y
Done
```

### Pattern Test a Range of Sectors

**p[attern test]** [*nnn* **times**] *cyl hd sec* [**to** *cyl hd sec*] [**with** *pat1* [*pat2...*]]

This command pattern tests the selected sectors repetitively. It is used to test part or all of the system portion of the disk (all cylinders before the diagnostic cylinder and relocated sectors). If the test is constrained to one track, the data is automatically saved and restored when the operation is completed. If more than one track is selected, a warning prompt is displayed and the operation can be terminated at that point to prevent permanent overwrite of data. The automatic save can be disabled with the **toggle_save** command. The argument [**to** *cyl hd sec*] can be substituted with the argument [**to** *end*].

Arguments:

- [*nnn* **times**]— Repetition count. possible responses include:
  - **1**—Lower bound value, default value.
  - **2,147,483,647**—Upper bound value.
- *cyl hd sec*— Starting sector.
- [**to** *cyl hd sec*] Ending sector, default is starting sector.
- [**with** *pat1*[*pat2...*]]— Up to 16 4-byte hex patterns can be specified. The default is the following seven patterns: `aaaaaaaa ffffffff ebd6ebd6 d7add7ad af5baf5b 5eb75eb7 6db66db6`

### Restore Previously Saved Track Data

**r[estore_track_data]**

This command can restore data from the general buffer to the original track (the track where the data was originally stored). The **save_track_data** command stores data from a selected track into the general buffer. In addition, the **format** and **pattern_test** commands save into the general buffer before their operation and automatically restore the data afterward. The data remains in the buffer so it can be restored again at a later time if, for instance, the first restore was bad.

If data is saved and not restored and then an attempt is made to quit the interactive test, a message is displayed indicating there is saved data and a prompt is displayed that allows reentering the interactive test or exiting the test.

The following figure shows an example of using the restore command:

## Figure dev4110–26, Restore Command Example

```
(D3;Save)-> restore
        Data has been saved for drive 3: cylinder 273  head 17
        Do you really want to restore this data? [yn] -> y
        Data restore operation complete
(D3;Save)->
```

If **n** is entered, the data is not restored, the operation is terminated, and the test returns to the interactive test prompt.

If the current drive has been changed from drive 3 to drive 9 since the last save operation, the following is displayed:

## Figure dev4110–27, Changing Drives Since Last Save Operation

```
(D9;Save)-> restore
        Data has been saved for drive 3: cylinder 273  head 17
        However, drive 9 is currently selected.
        Use 'd[rive_select]' to change drives and try again.
(D9;Save)->
```

If data was not saved before attempting to restore the data, the following is displayed:

## Figure dev4110–28, Attempting to Restore Data Before Saving

```
        No data has been previously saved so restore aborted.
(D9;Save)->
```

### Save One Track of Data

**sa[ve_track_data]** *cyl hd*

This command saves one track of data into the general buffer. The data can be restored with the **restore_track_data** command. Data on relocated sectors is also saved so, if the track is reformatted (which removes the relocated sectors), all the track's data can be restored.

> **NOTE**
>
> The prompt in the following figure specifies `Nosave`. This indicates the command **toggle_save** has previously been entered so saves are not automatic on destructive operations. It has no affect on this command. Normally, `Save` appears since disabling automatic save can be data destructive.

The following is an example of saving one track:

## Figure dev4110–29, Saving One Track

```
(D4;Nosave)-> save 525 5
        Data has already been saved for drive 4: cyl=601, hd=14
        Are you sure you want to overwrite this data? [yn] -> y
        Data save operation complete
(D4;Nosave)->
```

### Produce File of All Bad Sectors

**sc[an_flaws]** *filename*

This command reads every header on the disk looking for headers marked bad or which contain invalid information. Each of these headers produces an entry in the ASCII file entered on the command line. The file is in the same format as is read by the **file** command in Subtest 100, Input Defects Before Formatting. Refer to Subtest 100 for a description of the file format.

### Display Data From Range of Sectors

**se[ctor_display]** *cyl hd sec* [**to** *cyl hd sec*]

This command displays the data from one or more consecutive sectors.

The following figure shows an example of the **sector_display** command:

**Figure dev4110–30, Displaying Sector Data**

```
(D1;Save)-> sector 27 5 55

        Sector  27  5 55:
000  6db66db6 6db66db6 6db66db6 6db66db6  6db66db6 6db66db6 6db66db6 6db66db6
020  6db66db6 6db66db6 6db66db6 6db66db6  6db66db6 6db66db6 6db66db6 6db66db6
040  6db66db6 6db66db6 6db66db6 6db66db6  6db66db6 6db66db6 6db66db6 6db66db6
060  6db66db6 6db66db6 6db66db6 6db66db6  6db66db6 6db66db6 6db66db6 6db66db6
080  6db66db6 6db66db6 6db66db6 6db66db6  6db66db6 6db66db6 6db66db6 6db66db6
0a0  6db66db6 6db66db6 6db66db6 6db66db6  6db66db6 6db66db6 6db66db6 6db66db6
0c0  6db66db6 6db66db6 6db66db6 6db66db6  6db66db6 6db66db6 6db66db6 6db66db6
0e0  6db66db6 6db66db6 6db66db6 6db66db6  6db66db6 6db66db6 6db66db6 6db66db6
100  6db66db6 6db66db6 6db66db6 6db66db6  6db66db6 6db66db6 6db66db6 6db66db6
120  6db66db6 6db66db6 6db66db6 6db66db6  6db66db6 6db66db6 6db66db6 6db66db6
140  6db66db6 6db66db6 6db66db6 6db66db6  6db66db6 6db66db6 6db66db6 6db66db6
160  6db66db6 6db66db6 6db66db6 6db66db6  6db66db6 6db66db6 6db66db6 6db66db6
180  6db66db6 6db66db6 6db66db6 6db66db6  6db66db6 6db66db6 6db66db6 6db66db6
1a0  6db66db6 6db66db6 6db66db6 6db66db6  6db66db6 6db66db6 6db66db6 6db66db6
1c0  6db66db6 6db66db6 6db66db6 6db66db6  6db66db6 6db66db6 6db66db6 6db66db6
1e0  6db66db6 6db66db6 6db66db6 6db66db6  6db66db6 6db66db6 6db66db6 6db66db6

(D1;Save)->
```

This example is almost identical to the example for the **hde_display** command. The major difference in the two methods is that with the **sector_display** command, the read is sensitive to data errors. In the raw read mode that the **hde_display** command uses, no data errors are reported. If a data error does occur while executing the **sector_display** command, it is retried 10 times and any successful read results in the sector being displayed without an error message. If ten unsuccessful attempts to read the sector occur, then an error message is displayed and a prompt is displayed whether to retry, force, or skip the read.

To retry the read another 10 times press RETURN. To retry the read a different number of times, enter a number. From 1 to 2,147,483,647 retries can be specified. Every five retries a seek is alternately performed one cylinder in and back or one cylinder out and back.

If **force** is entered for the read, the buffer the data was supposed to be read into is displayed. If an ECC error occurred, the data is whatever was read. If some other error occurred, the data may be whatever was in the buffer before the read and be invalid. The error reported is the last error that occurred; it can be used to determine if the data is valid.

> **NOTE**
>
> At present, the software does not correct the data if it is a correctable ECC error.

If **skip** is entered, the sector is skipped. If there are more sectors to read in the range specified, then the display continues with the next sector.

The following is an example of a read that fails:

## Figure dev4110–31, Failed Read Example

```
(D1;Save)-> sector 400 22 18 to 400 22 10

dev4110 ERROR (0x06) ccu/mb/csr/d 3/0/3f0/1 - Hard ECC Error
 cyl:400 hd:22 sec:18

                        10 attempts to read sector failed
                        what to do [# of retries/force/skip] (10) -> 100

dev4110 ERROR (0x06) ccu/mb/csr/d 3/0/3f0/1 - Hard ECC Error
 cyl:400 hd:22 sec:18

                        100 attempts to read sector failed
                        what to do [# of retries/force/skip] (100) -> skip

        Sector 400 22 19
000  00000000 00000000 00000000 ff300020  1023eee10 99903220 b2d3d4d1 ab222290
020                     ------------ and so on -------------
```

Sector 18 was not displayed since it was skipped. What was displayed was the next sector, sector 19 which was read successful. The next sector (sector 19) was displayed because it was read successfully in one of the first 10 attempts.

### Slip One or More Sectors

**sl[ip_sectors]**

**CAUTION**

Read the caution and following example at the end of this command's description before using this command.

This command is used to slip sectors which are generating errors or to enter defect map information to **slip sectors**. If automatic save of track data is enabled (indicated by Save in the prompt), an attempt to save and restore all sectors is made to minimize data loss unless automatic save is disabled with the **toggle_save** command.

Entering the **slip_sectors** command starts a separate command processor which expects bad sectors or defect map information to be entered. A help facility which lists the available commands is displayed if **h** is entered. The help output appears as follows:

### Figure dev4110–32, *slip_sectors* Command Help Screen

```
SLIP_SECTORS commands:
c[hange_mode]          - toggles logical sector/defect map entry mode
cyl hd sec             - logical sector to be slipped
cyl hd bcai len        - defect map entry to be slipped
d[elete] cyl hd sec    - delete logical sector entry
d[elete] cyl hd bcai len - delete defect map entry
e[xecute]              - slip the sectors now
h[elp]                 - display this information
l[ist]                 - list inputs made so far
q[uit]                 - exit from 'slip_sectors'
!unix-command          - execute unix command
'comment               - comment; ignored
```

To slip sectors which are generating errors, the normal usage is to display the track headers for each track on which a sector is to be slipped by using the **track_headers_display** command. This command displays the headers before they are altered so the slip can be verified. Then enter all sectors which need to be slipped, verify the entries using the **list** command, delete any incorrect entries with the **delete** command, and then begin the slip with the **execute** command. New sectors can be entered at any time until the **execute** command is completed. As a safeguard, when the **execute** command is entered, the list of sectors is automatically displayed and a prompt is displayed to confirm the list. If **n** is entered, execution returns to the **slip_sectors** prompt with all inputs still intact.

To exit the test at any time enter the **quit** command.

To enter defect map information, first enter the **change_mode** command and then follow the above procedure. It may not be desirable to display track headers if there is a large number of tracks.

**NOTE**

When entering the defect map for a new disk, a disk that has just been formatted and does not yet contain a file system, the automatic save may be disabled with the **toggle_save** command before entering the **slip_sectors** command. This causes the slip to proceed much faster.

**CAUTION**

If the **toggle_save** command is executed prior to the slip, any usable data that exists on the disk will be lost, so use this approach with caution.

The current input mode (logical sector or defect map mode) is revealed in the slip prompt. If logical sector input mode is selected, the prompt appears as follows:

```
slip (D1;Save;Sector)->
```

If defect map input mode is selected, the prompt appears as follows:

```
slip (D1;Save;Defect)->
```

When a defect is entered, the associated track's headers are immediately read and a list of logical sectors that correspond to the defect are listed as shown in the following figure:

**Figure dev4110–33, Entering a Defect**

```
slip (D1;Save;Defect)-> 293 1 572 9510
        Associated logical sectors: *S 0 1
```

The previous figure illustrates that three sectors are affected by the media flaw which starts at byte 572 from index and is 9510 bits long. The *S indicates that one of the affected sectors is the spare sector. If a *B is displayed, then the media defect affects an already bad sector which is not reslipped.

The byte before and after any flaw is also considered bad to avoid the possibility that a flaw might be slightly larger than is reported in the defect map.

If two defects affect the same sector, the logical sector is only entered in the *INPUTS THAT WILL BE SLIPPED* input list once. Subsequent entries of the same sector are entered in the *INPUTS THAT WILL NOT BE SLIPPED* list. For example, the following two defects exist:

| cylinder | head | bcai | bit-length |
|---|---|---|---|
| 10 | 0 | 70 | 1 |
| 10 | 0 | 80 | 1 |

Both defects are in sector 0 so when the second defect is entered, the sector is not slipped again. The slipping of a sector with two defects is shown in the following figure:

**Figure dev4110–34, Slipping Sectors With Two Defects**

```
slip (D1;Save;Defect)-> 10 0 70 1
        Associated logical sectors: 0
slip (D1;Save;Defect)-> 10 0 80 1
        Associated logical sectors: (0)
        Sectors in () have already been input and will not result in a new
        entry. A (*G) means the flaw falls entirely in a gap and is ignored.
slip (D1;Save;Defect)-> list

 INPUTS THAT WILL NOT BE SLIPPED:
 CYL HD SEC   BCAI   LEN LOC  ERR SRC | CYL HD SEC   BCAI   LEN LOC  ERR SRC
  10  0   0     80     1       N/A USR |

 INPUTS THAT WILL BE SLIPPED:
 CYL HD SEC   BCAI   LEN LOC  ERR SRC | CYL HD SEC   BCAI   LEN LOC  ERR SRC
  10  0   0     70     1       N/A USR |
```

The above example also refers to a *gap* which is listed in the Associated logical sectors list as a (*G). The *G indicates that the entire media flaw was in a space between sectors or located after the last sector on the track.

> **NOTE**
>
> In the current version of the *dev4110* test, no bytes are considered part of a gap. A flaw in any byte results in one or more sectors being slipped.

When the slip is executed, the sectors are slipped one track at a time. If automatic save is active, the data for the track is copied to a special buffer (not the general buffer used by the **save_track_data** command) before the slip and is restored after the slip. One problem exists which is that correctable ECC errors are not corrected so a correctable ECC error may not be fixable without a loss of data.

To verify the slip, display the track headers once more. All slipped sectors are marked with a ***B**.

An example of slipping the sectors 17 7 23, 17 7 36, 247 12 4, and 255 0 12 is shown in the following figure:

**NOTE**

Track 255 already has a slipped sector. Sector 12 that is about to be slipped *must* be a new error since sector 6 was slipped. Refer to the caution and following example at the end of the **slip_sectors** command description for more information about slipping two or more sectors on one track.

## Figure dev4110–35, Display Slipped Track Headers

```
(D1;Save)-> track 17 7
cylinder =  17  head =  7
53 54 55 56 57 58 *S  0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17
18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42
43 44 45 46 47 48 49 50 51 52
(D1;Save)-> track 247 12
cylinder = 247  head = 12
48 49 50 51 52 53 54 55 56 57 58 *S  0  1  2  3  4  5  6  7  8  9 10 11 12
13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37
38 39 40 41 42 43 44 45 46 47
(D1;Save)-> track 255 0
cylinder = 255  head =  0
 0  1  2  3  4  5 *B  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23
24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48
49 50 51 52 53 54 55 56 57 58
```

If defect input mode is selected and an attempt is made to slip the sector on track 255 head 0 which has already been marked bad in the previous figure, the following figure shows what would be displayed:

## Figure dev4110–36, Slipped Tracks Previously Marked Bad

```
slip (D1;Save;Defect)-> 255 0 3800 1
        Associated logical sectors: (*B-6i)
```

The associated logical sector has been previously marked bad (*B). The 6i indicates the sector's location from index. Starting with the first sector after index as 0, the sector marked bad is sector 6 after index (which is actually the seventh sector after index). This notation is necessary because, once a header is marked bad, the header no longer contains a logical sector number. The only known information is the header's position from index.

The following illustration shows how to delete incorrect **slip_sectors** inputs:

## Figure dev4110–37, Deleting Incorrect *slip_sectors* Inputs

```
(D1;Save)-> slip
Type 'h' for help with slip commands
slip (D1;Save;Sector)-> 17 7 23
slip (D1;Save;Sector)-> 17 7 46
slip (D1;Save;Sector)-> d 17 7 46
        Sector  17  7 46 has been removed from the inputs
slip (D1;Save;Sector)-> list

 INPUTS THAT WILL NOT BE SLIPPED:
 CYL HD SEC   BCAI   LEN LOC  ERR SRC | CYL HD SEC   BCAI   LEN LOC  ERR SRC
         ---------------------- no sectors ----------------------

 INPUTS THAT WILL BE SLIPPED:
 CYL HD SEC   BCAI   LEN LOC  ERR SRC | CYL HD SEC   BCAI   LEN LOC  ERR SRC
  17  7  23                   N/A USR |


slip (D1;Save;Sector)-> 17 7 36
slip (D1;Save;Sector)-> 247 12 4
slip (D1;Save;Sector)-> 255 0 12
slip (D1;Save;Sector)-> e

 INPUTS THAT WILL NOT BE SLIPPED:
 CYL HD SEC   BCAI   LEN LOC  ERR SRC | CYL HD SEC   BCAI   LEN LOC  ERR SRC
         ---------------------- no sectors ----------------------

 INPUTS THAT WILL BE SLIPPED:
 CYL HD SEC   BCAI   LEN LOC  ERR SRC | CYL HD SEC   BCAI   LEN LOC  ERR SRC
  17  7  23                   N/A USR | 247 12   4                   N/A USR
  17  7  36                   N/A USR | 255  0  12                   N/A USR

        Are you sure inputs are ready for slipping? [yn] -> y

      sector  17  7 23 and
              17  7 36: slipped
      sector 247 12  4:
dev4110 ERROR (0x06) ccu/mb/csr/d 3/0/3f0/1 - Hard ECC error
 cyl:247 hd:12 sec:4

                        10 attempts to read sector failed
                        what to do [# of retries/force/skip] (10) -> 100

slipped
        sector 255  0 12: slipped
```

Sector 247 12 4 was difficult to read. After the first 10 attempts failed, a request was entered to retry 100 times. One of the 100 retries was successful so the sector slip operation was completed. If the 100 retries had been unsuccessful, a prompt to retry again, force continuation with the data that had been read from sector 4, or skip slipping this sector would have been displayed. If **skip** is selected, the remaining slips are still performed. Every five retries a seek is alternately performed one cylinder in and back or one cylinder out and back.

To verify the slips, display the track headers again as shown in the following figure:

## Figure dev4110–38, Displaying the Track Headers

```
(D1;Save)-> track 17 7
cylinder =  17  head =  7
52 53 54 55 56 57 58  0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17
18 19 20 21 22 *B 23 24 25 26 27 28 29 30 31 32 33 34 *B 36 37 38 39 40 41
42 43 44 45 46 47 48 49 50 51
(D1;Save)-> track 247 12
cylinder = 247  head = 12
47 48 49 50 51 52 53 54 55 56 57 58  0  1  2  3 *B  4  5  6  7  8  9 10 11
12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36
37 38 39 40 41 42 43 44 45 46
(D1;Save)-> track 255 0
cylinder = 255  head =  0
 0  1  2  3  4  5 *B  6  7  8  9 10 11 *B 13 14 15 16 17 18 19 20 21 22 23
24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48
49 50 51 52 53 54 55 56 57 58
(D1;Save)->
```

Track 17 7 and track 255 0 now have only 58 logical sectors because one sector has been relocated from each track. The Bad Block Table may be displayed as shown in the following figure to verify the relocation actually occurred:

## Figure dev4110–39, Displaying the Bad Block Table

```
(D1;Save)-> bbt

BAD BLOCK TABLE FOR ccu/mb/csr/d=3/0/3f0/0

         CYL HD SEC  CYL HD SEC  CYL HD SEC  CYL HD SEC  CYL HD SEC
          17  7  35  255  0  12

(D1;Save)->
```

If there had already been entries in the table, they would have been displayed also. This list of the Bad Block Table is a copy of the table in main memory. However, it does reflect what is on the disk since both the **slip_sectors** and **format** commands rewrite all five copies of the Bad Block Table before returning to the interactive prompt.

Execute the **verify_format** command for each of the tracks to verify the tracks.

**CAUTION**

The **slip_sectors** command has an inherent danger. If logical sectors are input to be slipped and two or more sectors are to be slipped on the same track, all the sectors that are on the same track must be slipped at one time since the entered sector numbers that are given to **slip_sectors** are logical numbers. **Slip_sectors** searches the headers on the track to find the sectors to slip. If a slip of one sector on a track is executed and then later a slip of another sector is executed (without reverifying the logical sector number of this second sector), the wrong sector might be slipped.

The following is an example to illustrate the preceding caution.

Cylinder 25, head 0 in sectors 8 and 12 is generating correctable ECC errors; therefore, the **track** command is executed as shown in the following figure to display the headers before the slip is performed:

## Figure dev4110–40, Display Track Headers Prior to Slipping

```
(D1;Save)-> track 25 0
cylinder =  25  head =  0
 0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24
25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49
50 51 52 53 54 55 56 57 58 *S
```

The following figure shows the headers that are displayed if only sector 8 is slipped:

## Figure dev4110–41, Display Slipped Track Headers

```
(D1;Save)-> track 25 0
cylinder =  25  head =  0
 0  1  2  3  4  5  6  7 *B  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23
24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48
49 50 51 52 53 54 55 56 57 58
```

Sector 12 is no longer in the same place. The remaining bad sector has 11 as a header. If the **slip_sectors** command is reentered and sector 12 is slipped, correctable ECC errors will probably occur on sector 11 in the future.

If only sector 8 is slipped, the following two options are available:

1. Reformat the track which unslips sector 8 and then slip both sectors at one time. Display the track headers before the slip to verify only sector 8 has been previously slipped. If other sectors have been unslipped on this track in the past, slip them also.
2. Check the track headers and then slip the sector which is now in sector 12's place. In this example, slip sector 11.

If sector 12 has been incorrectly slipped, unslip the bad slip by performing option 1.

### Display Current Drive and Saved Track

**st[atus]**

This command lists the Drive Configuration Data (for all selected drives) and displays which drive is the active drive. The **Status** command also displays which track was last saved in the general buffer by one of the commands: **format**, **save_track_data**, or **pattern_test**. In addition, a line is displayed which indicates whether automatic save of track data is enabled (Save is also shown in the prompt).

The following figure shows the information that is displayed when the **status** command is entered:

**Figure dev4110–42, *status* Command**

```
(D1;Save)-> status

                    DRIVE CONFIGURATION DATA

    IOP MBus MBus  Int   Unit  #    #  Phys  Log
     #   #   CSR  Level   #   Cyl  Hds  Sec  Sec  Name
    --- --- ----- -----  ---  ---  ---  ---  ---  -------
 1.  3   0  0x3f0   2     0   760   19   60   59  DKD-005
 2.  3   0  0x3f0   2     1   760   19   60   59  DKD-005
 3.  3   0  0x3f8   3     0   842   20   46   45  DKD-001

        Drive 1 is currently selected

Saved Track:  drive 3  cylinder 300 head 12

Save of track data is automatic during destructive commands.
(D1;Save)->
```

### Switch Between Enable and Disable of Automatic Save

**to[ggle_save]**

This command switches between enable and disable of automatic save. Automatic save occurs during the commands **format**, **pattern_test**, and **slip_sectors**. The displayed interactive prompt indicates whether automatic save is enabled (`Save`) or disabled (`Nosave`). Also, the **Status** command displays the current setting of automatic save.

### Display Sector Numbers

**tr[ack_headers_display]** *cyl hd* [**to** *cyl hd*]

This command displays the sector numbers for each of the sectors on a track. The first sector number is located physically just after the index so the number is not always zero since sectors are skewed depending on the head number. For example, on head 0, logical sector 0 is the first sector after index, and on head 1, logical sector 0 is the second sector.

Each track has one spare sector (unless it is used because one or more sectors were slipped on the track). The spare sector is marked ***S**. If any sectors have been flagged as bad, they are marked ***B**.

The following figure shows the display of sector numbers for the sectors and a ***S** for a spare sector on a track:

**Figure dev4110–43, Displaying Sector Numbers and Spare Sector**

```
(D1;Save)-> track 427 2
cylinder = 427  head =  2
58 *S  0  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22
23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47
48 49 50 51 52 53 54 55 56 57
(D1;Save)->
```

In the previous figure, sector 0 is actually the third sector on the track since it is head 2. Also the spare sector (***S**) is just before sector 0.

The following figure shows the display of sector numbers for the sectors and a ***B** for a bad sector on a track:

## Figure dev4110–44, Displaying Sector Numbers and Bad Sector

```
(D1;Save)-> track 208 12
cylinder = 208  head = 12
47 48 49 50 51 52 53 54 55 56 57 58  0  1  2  3  4  5 *B  6  7  8  9 10 11
12 13 14 15 16 17 18 19 20 21 22 *B 24 25 26 27 28 *B 30 31 32 33 34 35 36
37 38 39 40 41 42 43 44 45 46
(D1;Save)->
```

In previous figure, three sectors have been flagged bad. The spare sector has been used and sectors 23 and 29 have been relocated and appear in the Bad Block Table (refer to the **bad_block_table** command for more information). Also sector 0 is the 13th sector on the track because this is head 12.

### Read a Range of Sectors

**v[erify_format]** [*nnn* **times**] *cyl hd sec* [**to** *cyl hd sec*]

This command allows repetitive reads of a range of sectors to attempt to detect read errors. The command is not data destructive so it can be used on a customer's disks. It does not perform a data compare since the data is unknown. As the verify progresses, a pass count is incremented on the display to indicate how far the execution has progressed. Any errors are displayed without retry and the reads continue. If any sectors have been relocated, the relocated sectors are read. The argument [**toR** *cyl hd sec*] can be substituted with [**to** *end*]

### Execute UNIX Command

! *UNIX-command*

This command is used to issue UNIX commands. After the UNIX command is completed, execution is returned to the interactive prompt.

The following figure shows an example of entering the ! interactive test command and the **ps** UNIX command:

**Figure dev4110–45, Issuing a UNIX Command**

```
(D1;Save)-> !ps
   PID TTY TIME CMD
     2 co  0:08 -
   273 co  0:00 dshell
   274 co  0:09 dev4110.t -c 2
   281 co  0:00 sh -c ps?
   282 co  0:02 ps
(D1;Save)->
```

### Enter Comments

*' comment*

This command is used to enter a comment. To enter a comment, type the ' interactive test command followed by a comment.

# Disk Parameters File, *DB_diskfmt* Description

The disk parameters file, */mnt/bin/lib/DB_diskfmt*, contains information about disk drives. This information is required to be able to format or test new drives. Each line in the file contains a number of fields separated by one or more spaces. Comments start with a # and continue to the end of the line. Blank lines are also allowed. To add new drives, create a new entry with all the fields defined, and then add the drive to the */ioconfig* file. As of the time of this printing, *DB_diskfmt* contains the following information for all CONVEX machines:

## Figure dev4110–46, Contents of the *DB_diskfmt* File

```
# DB_diskfmt - file of disk parameters
# >>>> WARNING - DO NOT USE 'diskfmt' TO FORMAT! It is no longer compatible
# >>>>           with the CONVEX FORMAT! Instead, format MBUS-attached drives
# >>>>           with 'dev4110' and VME-attached drives with 'dev5130'.
# KEY FOR DRIVE NAMES (unformatted capacity is given in parentheses):
#  Name          Description           Name          Description
# DKD-001       Fujitsu Eagle (452MB) DKD-008,208  NEC 2363 (1080MB)
# DKD-002       CDC 9766 (300MB)      DKD-214      Hitachi DK514-38 (356MB)
# DKD-005,206   NEC 2352 (500MB)
#------- XYLOGICS 450/451 SMD CONTROLLER (MBUS) -------
#  a    b   c   d  e  f   g    h    i j k  l  m   n    o  p
DKD-001 2  842 20 46 45 4800 28160 1 0 1  0  0  smd mfm y
DKD-002 0  823 19 32 31 5040 20160 1 0 1  0  0  smd mfm y
DKD-005 0  760 19 60 59 4832 36288 1 0 1  0  0  smd 2-7 y
DKD-008 1 1024 27 68 67 4816 40960 1 0 1  0  0  smd 2-7 y
#------- INTERPHASE 4201 ESDI CONTROLLER (VME) --------
#  a    b   c   d  e  f   g    h    i j k  l  m   n    o  p
DKD-214 0  903 14 51 50 4736 30240 5 5 1  8  8 esdi 2-7 n
#------- INTERPHASE 4200 SMD CONTROLLER (VME) ---------
#  a    b   c   d  e  f   g    h    i j k  l  m   n    o  p
DKD-206 0  760 19 60 59 4832 36288 5 4 1 12 12  smd 2-7 n
DKD-208 0 1024 27 68 67 4816 40960 5 6 1 16 12  smd 2-7 n
# LEGEND:
# a - drive name                Must be DKD-0XX for Multibus and DKD-2XX for
#                               VMEbus
# b - disk type                 For Xylogics controller
# c - # of cylinders
# d - # of heads
# e - # of physical sectors     Number of actual sectors excluding runt
# f - # of logical sectors      Number of physical sectors (e) minus number of
#                               spares
# g - bits per sector           Number of bits between sector pulses
# h - bytes per track           Total number of unformatted bytes per track
# i - skew                      Sector offset from one head to the next
#                               Must be 1 when using Xylogics 450/451 cntlr
# j - # of relocation tracks    .5% of number of cylinders (c). Raise
#                               fractional part to next higher whole number.
#                               Ignored by dev4110 (Multibus formatter)
# k - interleave                sector separation between consecutive sectors
#                               Currently must be 1 for Xylogics 450/451 cntlr
# l - gap 1 size                Number of halfwords in gap before header
#                               (2 bytes per halfword)
#                               Ignored by dev4110 (Multibus formatter)
# m - gap 2 size                Number of halfwords in gap following header
#                               (2 bytes per halfword)
#                               Ignored by dev4110 (Multibus formatter)
# n - drive interface           Used to determine how to read manufacturer's
#                               defect map.  Currently, smd or esdi
#                               Ignored by dev4110 (Multibus formatter)
# o - data encoding scheme      Way data is encoded on the media. Used to
#                               select patterns for pattern test.
#                               Currently mfm, 2-7 or 1-7
# p - Are spares interleaved    For Xylogics, 'y'. For Interphase, 'n'.
```

# Diagnostic Cylinder Description

The diagnostic cylinder is located on the first full cylinder preceding the sector forwarding area. Cylinder 840 is the diagnostic cylinder on a Fujitsu Eagle drive, and cylinder 820 is the diagnostic cylinder on a CDC SMD drive.

Each track of the cylinder is independent of the other tracks since there is a "table of contents" stored at sector 0 of each track This table describes what is stored on the remainder of that track. The format of this table is shown below:

**Figure dev4110–47, Diagnostic Cylinder Table of Contents Format**

```
/* ------------------------------------------------------------------- *
 *              Diagnostic Cylinder Definitions                        *
 * ------------------------------------------------------------------- */
#define NOT_USED        0x00000000        /* position in tbl not used     */
#define NO_HDR          0x00000001        /* this sectors header deleted  */
#define TBL_CONT        0x00000002        /* sector contains tbl of cntnts*/
#define ECC_ERR         0x00000003        /* sector written with bad ecc  */
#define PATTERN         0x00000004        /* sector written with pattern  */

 struct   tbl_cont {
    int   magic_nbr;            /* identifies this as a table of contents */
    int   version;              /* version number of this table */
    int   cyltksc;              /* sector header for this sector*/
    struct {
        int   sec_cont;         /* defines contents of corresponding sector */
        int   pattern;          /* pattern or mask used to verify contents  */
    } sec_tbl[62];
    int   checksum;             /* arithmetic tally of all the above fields */
 };
```

Each track on the cylinder is formatted in such a way that sector 0 is always on the track; sector 0 is written and verified without error when the drive is formatted.

The first field in the table contains the magic number 0x84101500. If this number is not present in the first position of a table, then someone has altered the diagnostic cylinder.

The second field in the table contains the version number. This number is an ordinal number starting with one and increasing as needed. Its purpose is to allow some measure of compatibility of newer diagnostic cylinders with older software. This number is checked if an unknown entry in the sector table (sec_tbl) is encountered. If the version number in the table is greater than the version number of the software, the diagnostic cylinder is assumed to have been formatted with a newer revision of software, and the opcode in the sector table is new to the older software. Therefore, the software is not familiar with the opcode, and the entry in the sec_tbl is ignored.

The next field in the table contains a copy of the sector header (cyltksc) for the table of contents.

The next field in the table contains the sector table (sec_tbl). This table describes what is stored on the rest of the sectors in this track. The sector table consists of 62 entries of 2 int's. Each entry (0 - 61) in this table, corresponds to a sector (0 - 61) on the track. The first position in each entry is the sector contents (sec_cont) field. If the track has more than 62 sectors, the extra sectors are not used or checked by current software. The following table describes what is currently defined:

**Table dev4110–9, Defined Values for Sector Contents Field**

| FIELD | DESCRIPTION |
|---|---|
| NOT_USED | This sector has not been initialized and should not be checked. |
| NO_HDR | When this sector is read, the controller should return a 0x05 sector not found error. |
| TBL_CONT | This sector contains a copy of the table of contents. |
| ECC_ERROR | This sector contains a copy of the table of contents that has been corrupted to give some form of ECC error. The pattern field contains a mask which may be xor'ed with the magic number field to correct the error. If the mask contains 11 or fewer one bits, the controller should report a soft ECC error when this sector is read. If the mask contains more than 11 one bits, the error reported should be a hard ECC error. |
| PATTERN | This sector has been filled with the data contained in the pattern field. No errors should be encountered when the sector is read. |

The last field in the table of contents is a checksum. This is an arithmetic tally of all the bytes in the table of contents. This value is checked before any of the data in the table of contents is used.

# Error Codes

The *dev4110* test reports controller and drive errors in a standard format. However, if additional data is available at the time of the error, it reports that data also. The basic format for the errors is:

```
dev4110 ERROR (Err) ccu/mb/csr/d=c/m/rrr/d Err_desc
```

where:

| | |
|---|---|
| (*Err*) | Identifies the error code associated with the description. |
| *Err_desc* | Describes the type of error that occurred. |
| *c* | Gives the CCU slot number for the IOP. |
| *m* | Is the Multibus chassis number. |

*rrr* Gives the control and status register address (CSR), which identifies the board.

*d* Specifies the drive number. Each drive begins at 0 with additional drives of the same type being numbered 1, 2, and 3. The number 4 indicates the error occurred while performing a Set Drive Size or NOP command.

If no drive is associated with the error, then the controller and drive information does not appear in the error message.

Additional data may also be reported on a second line (depending on the type of error). For example, the format for data compare errors is:

```
Cyl:dddd Hd:dd
```

The following format appears when errors involve *write, read, read* and *write HDE, read* and *write track headers,* or *seek* commands (depending on the error, the sector number or number of seeks may not appear in the error message):

```
Cyl:dddd Hd:dd Sect:ddd Disp:ddd Exp:hhhh Act:hhhh #Seeks:d
```

where:

*Cyl* Specifies the position of heads on the drive.

*Hd* Identifies the head selected.

*Disp* Identifies number of bytes from the beginning of the sector (data compare errors only).

*Exp* Specifies the expected value (data compares errors only).

*Act* Specifies the actual value (data compares errors only).

*Sect* Indicates the sector where the error occurred.

*# Seeks* Identifies the number of seeks executed.

# Error Messages

Both *dev4100* and *dev4110* generate the same error codes and report errors in the same format. For details on common error codes and their descriptions, refer to the section entitled **Error messages** in the *dev4100* test description.

## Special Error Messages for *dev4110*

In some circumstances, a certain set of error messages are displayed if the peripheral is not powered on, is write protected, or is cabled incorrectly. The test gives the user several options to proceed with, depending on the circumstances. The following examples illustrate these type of error messages and user options that are presented. The first example

illustrates the messages produced when the peripheral is write protected and an operation attempts a write. The second example illustrates the messages produced when the peripheral is not ready (ie., power is not on or a cable is not connected).

> **NOTE**
>
> In the following examples, all **boldface** entries are user entered information.

## Figure dev4110–48, Write Protect Error Example

```
(D1;NoSave)-> pattern_test 840 0 30 with aaaaaaaa
     Patterns used:
       aaaaaaaa
pass # 1
dev4110 ERROR (0x14)-ccu/mb/csr/d=3/0/3f0/0 Write-protected disk
  Cyl: 840 Hd: 0 Sect: 30 # Seeks:1
  IOPB: cmd stat  drv hd/s  cyl  cnt addr relo ofst next  ecc eoff
       d100 8514 04a0 001e 4803 0100 0020 0010 0000 0000 0000 0000
Attempt to fix the problem.  You can issue UNIX commands if it helps.
Then enter one of the following:
        <CR> to retry the operation
        'nofix' if unable to fix the problem
UNIX command, <CR> for retry, or nofix: RETURN

dev4110 ERROR (0x14)-ccu/mb/csr/d=3/0/3f0/0 Write-protected disk
  Cyl: 840 Hd: 0 Sect: 30 # Seeks:1
  IOPB: cmd stat  drv hd/s  cyl  cnt addr relo ofst next  ecc eoff
       d100 8514 04a0 001e 4803 0100 0020 0010 0000 0000 0000 0000
Attempt to fix the problem.  You can issue UNIX commands if it helps.
Then enter one of the following:
        <CR> to retry the operation
        'nofix' if unable to fix the problem
UNIX command, <CR> for retry, or nofix: nofix
```

## Figure dev4110–49, Drive Not Ready Error Example

```
(D2;Nosave)-> verify_format 0 0 0
             Drive 2: Read of copy #1 of BBT from cyl:759 hd:18 sec:34 failed

dev4110 ERROR (0x16)-ccu/mb/csr/d=7/1/3f0/0 Drive is not ready or faulted
  Cyl: 759 hd:18 Sect: 34
IOPB: cmd stat  drv hd/s  cyl  cnt addr relo ofst next  ecc eoff
     d200 8516 0420 1222 f702 0500 1410 0050 0000 0000 0000 0000
      Drive 2: Read of copy #2 of BBT from cyl:759 hd:18 sec:39 failed

           .
           .
           .

dev4110 ERROR (0x16)-ccu/mb/csr/d=7/1/3f0/0 Drive is not ready or faulted
  Cyl: 759 hd:18 Sect: 49
IOPB: cmd stat  drv hd/s  cyl  cnt addr relo ofst next  ecc eoff
     d200 8516 0420 1231 f702 0500 1410 0050 0000 0000 0000 0000
      Drive 2: Read of copy #5 of BBT from cyl:759 hd:18 sec:54 failed

dev4110 ERROR (0x16)-ccu/mb/csr/d=7/1/3f0/0 Drive is not ready or faulted
  Cyl: 759 hd:18 Sect: 54
IOPB: cmd stat  drv hd/s  cyl  cnt addr relo ofst next  ecc eoff
     d200 8516 0420 1236 f702 0500 1410 0050 0000 0000 0000 0000

dev4110 ERROR (0x4d)-ccu/mb/csr/d=7/1/3f0/0 All copies of BBT are Bad
The internal copy of the BBT is cleared since no good copies were found
Do you want to retry the last operation [yn] -> n
verify_format: pass # 1
dev4110 ERROR (0x4d)-ccu/mb/csr/d=7/1/3f0/0 Drive is not ready or faulted
  Cyl:   0 Hd: 0 Sect:  0 # Seeks:1
  IOPB: cmd stat  drv hd/s  cyl  cnt addr relo ofst next  ecc eoff
       d200 8516 0420 0000 0000 0100 1410 0050 0000 0000 0000 0000

dev4110 ERROR (0x16)-ccu/mb/csr/d=7/1/3f0/0 Drive is not ready or faulted
  Cyl:   0 Hd: 0 Sect:  0 # Seeks:1
  IOPB: cmd stat   drv hd/s  cyl  cnt addr relo ofst next  ecc eoff
Attempt to fix the problem. You can issue UNIX command if it helps.
Then enter one of the following:
        <CR> to retry the operation
        'nofix' if unable to fix the problem
UNIX command, ,CR> for retry, or nofix: RETURN

dev4110 ERROR (0x16)-ccu/mb/csr/d=7/1/3f0/0 Drive is not ready or faulted
  Cyl:   0 Hd: 0 Sect:  0 # Seeks:1
  IOPB: cmd stat   drv hd/s  cyl  cnt addr relo ofst next  ecc eoff
Attempt to fix the problem. You can issue UNIX command if it helps.
Then enter one of the following:
        <CR> to retry the operation
        'nofix' if unable to fix the problem
UNIX command, ,CR> for retry, or nofix: noflx
```

# dev4200

# Multibus STC Tape Unit Controller Test

## Overview

The *dev4200* test verifies the functionality of CONVEX Multibus STC tape controllers.

## Prerequisites and Required Equipment

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

**Table dev4200–1, Hardware Requirements**

| C1, C120 | C200 Series |
|---|---|
| MCU | Memory System[1] |
| MAU | CPX |
| SPU | SP2 |
| IOP | IOP |
| MBCU | PIA |
| MBTC[2] | MBCU |
| Tape Drive | MBTC[2] |
| | Tape Drive |

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

[2] MBTC represents Multibus Tape Controller

## Test Invocation

The *dev4200* 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 *dev4200* 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 dev4200–1, Test Invocation Sequence

```
(spu)> cd /mnt/test RETURN
(spu)> sysreset RETURN
(spu)> mminit -s RETURN
(spu)> dshell RETURN
: test dev4200 [-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 dev4200** executes all *dev4200* 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 powerup. 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 if 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 dev4200–2, Alternate Test Invocation Sequence

```
(spu)> cd /mnt/test RETURN
(spu)> initall RETURN
(spu)> dshell RETURN
: test dev4200 [-c [class number(s)]] [-s [subtest number(s)]] [+> filename] RETURN
```

## 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 dev4200–2, Getting Help During Test Parameter Entry

| Character | Description |
|---|---|
| ? | Displays this help menu |
| h | Provides help for a specific prompt |
| i | Displays the */ioconfig* file |

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

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

## Figure dev4200–3, 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

            PERIPHERAL CONFIGURATION DATA
      CCU       Chassis  Type    CSR     Int     Unit    Type
      ------    -------  ------  -----   ---     ----    -------
 1)   iop 3       0      MTC-001 0x0c0    4        0     MTD-001
 2)   iop 3       0      MTC-001 0x0c0    4        0     MTD-001

Device 0 = user defined configuration

 1: Device Selection [0,1-2][1]                          (0) ->
 2: IOP [3-7][2]                                        (5) ->
 3: Multibus Chassis [0-3]                              (0) ->
 4: Controller Offset in Multibus [0x0-0xfff]
                                                    (0xc0) ->
 5: Controller Interrupt [0-7]                          (4) ->
 6: Tape Unit [0-3]                                     (0) ->

        Tape Unit Types:
        0 -> STC 2920
        1 -> STC 1960
        2 -> FUJITSU M2436
        3 -> CDC 92185-04

 7: Tape Unit Type [0-3]                                (0) ->
 8: Media Error Retry Count in Subtests n50-n53 [0-15]
                                                        (5) ->
 9: Display Each Media Error [y,n]                      (n) ->
10: Run Subtests n50-n53 to EOT [y,n]                   (n) ->
11: Fixed Length Block Size, Subtests n10-n12,n30,n50,n51
    [18-131071]                                     (512) ->
12: Run EOT Subtest n25 [y,n]                          (n) ->
13: Allow Manual Intervention [y,n]                    (n) ->
14: Enter OK, or :NN to return to question NN [OK]
                                                       (OK) ->
```

[1] The options for this prompt change depending on how many tape units are in the system configuration.

[2] The options for this prompt change depending on the machine architecture. For C1 and C120 machines, the available selection ranges from 3-7. For C200 Series machines, the available selection ranges from 0-3.

> **NOTE**
>
> In prompts 8, 10, 11, and 12 in the previous figure, *n* varies from 3 to 5 depending on which classes are executed. Class 3 subtests execute at 800 bpi, Class 4 subtests execute at 1600 bpi, and Class 5 subtests execute at 6250 bpi. Refer to class and subtest descriptions for more information.

At any time during the test parameter sequence, several options are available as denoted at the top of the Test Parmeter 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

**?** — Request help for the current prompt (if available)

**^** — Return to the previous prompt

## Prompt Explanations

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

```
Device Selection [0,1-2]                    (0) ->
```

This prompt selects a device for testing from the system configuration file (*/ioconfig*). If **0** (default) is entered, then user-defined configuration prompts (2-5 in the previous figure) are displayed to determine the configuration to be tested. If a specific device is to be selected, the number of the device is entered (**1**, **2**, etc...). At that point, the additional configuration prompts (2-5 in the previous figure) are not displayed.

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

Enter the CCU slot number for the desired IOP.

```
Multibus Chassis [0-3]                      (0) ->
```

Enter the number of the chassis to be tested. (The drive is attached to a controller which is in a Multibus chassis.)

```
Controller Offset in Multibus [0x0-0xfff]
                                           (0xc0) ->
```

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

```
Controller Interrupt [0-7]                  (4) ->
```

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

```
Tape Unit [0-3]                                    (0) ->
```

Enter the unit number of the tape unit to be tested.

```
Tape Unit Type [0-3]                               (0) ->
```

Enter the number of the type of tape unit to be tested.

```
Media Error Retry Count in Subtests n50-n53 [0-15]
                                                   (5) ->
```

Enter the number of retries to be performed when a media error is detected during Subtests n50–n53.

```
Display Each Media Error [y,n]                     (n) ->
```

Enter **n** or RETURN so media errors and retries are not displayed during all subtests; otherwise, enter **y** to display all media errors and retries during all subtests.

```
Run Subtests n50-n53 to EOT [y,n]                (n) ->
```

Enter **y** to write as many files of data as are required to the end of the tape during Subtests n50–n53.

```
Fixed Length Block Size, Subtests n10-n12,n30,n50,n51
[18-131071]                                      (512) ->
```

Enter (RETURN) to select 512 bytes as the block size of records written to the tape during Subtests n10–n12, n30, n50, n51; otherwise, enter the block size (in bytes) of records written to the tape.

```
Run EOT Subtest n25 [y,n]                        (n) ->
```

Enter **n** or (RETURN) to skip Subtest n25; otherwise, enter **y** to allow execution of Subtest n25, End-of-Tape Sensing.

```
Allow Manual Intervention [y,n]                  (n) ->
```

Enter **n** to skip Subtests 104, 105, 200–202; otherwise, enter **y** to allow execution of Subtests 104, 105, 200–202.

```
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 dev4200–4, Sample Test Parameter Summary**

```
                TEST PARAMETER SUMMARY

Device Selection                                    : 0
IOP                                                 : 5
Multibus Chassis                                    : 0
Controller Offset in Multibus                       : 0xc0
Controller Interrupt                                : 4
Tape Unit                                           : 0

        Tape Unit Types:

             0 -> STC 2920
             1 -> STC 1960
             2 -> FUJITSU M2436
             3 -> CDC 92185-04

Tape Unit Type                                      : 0
Media Error Retry Count in Subtests n50-n53         : 5
Display Each Media Error                            : n
Run Subtest n50-n53 to EOT                          : n
Fixed Length Block Size, Subtests n10-n12,n30,n50,n51
                                                    : 512
Run EOT Subtest n25                                 : n
Allow Manual Intervention                           : n
Enter OK, or :NN to return to question NN          : OK
```

# 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.

# Class Descriptions

The *dev4200* test contains the five classes of subtests listed in the following table:

**Table dev4200–3, *dev4200* Test Classes**

| CLASS | DESCRIPTION |
|---|---|
| 1 | Controller loopback tests |
| 2 | Operator controls tests |
| 3 | Functional tests—800 bpi |
| 4 | Functional tests—1600 bpi |
| 5 | Functional tests—6250 bpi |

All of the subtests contained in *dev4200* are loopable under the *dshell*.

## Class 1 Subtests

Class 1 subtests verify the basic functionality of the controller. The subtests verify board reset and loopback capabilities. By entering **n** (the default) for the `Allow Manual Intervention` test parameter prompt, Subtests 104 and 105, which require manual intervention are not executed.

**Table dev4200–4, Class 1 Subtests**

| SUBTEST | DESCRIPTION | TIME (min:sec) |
|---|---|---|
| 100 | MBTC Reset Capability | 00:01 |
| 101 | MBTC Interrupt Loopback | 00:05 |
| 102 | MBTC Data Loopback | 00:25 |
| 103 | MBTC Memory Access | 04:30 |
| 104 | MBTC Command Loopback | 00:10 |
| 105 | MBTC Chain Command Loopback | 00:10 |

The execution times in the previous table are approximate times for testing a STC 2920 drive at 1600 BPI. Other drives and densities will affect these figures. Specifying a block size larger than the default changes the runtimes for these subtests. All times include rewind time from the tape position at the end of the last subtest.

The *dev4200* test does not test the drive commands: *DMS*, *SNS*, and *LWR*. In addition, it does not test the following controller-error handling capabilities:

- Pending command abort
- Multibus timeout

Also, certain drive capabilities are not tested. This includes all command reject and abort scenarios.

### Subtest 100, MBTC Reset Capability

Subtest 100 verifies that the controller resets properly by checking the status register for the correct value.

### Subtest 101, MBTC Interrupt Loopback

Subtest 101 verifies that the board interrupt capability can be turned off and on.

### Subtest 102, MBTC Data Loopback

Subtest 102 tests the direct memory access First-In-First-Out (FIFO) memory on the board. It pattern checks all RAM's in the FIFO and tests in/out capability, assuming the FIFO operates like a circular buffer with pointers.

### Subtest 103, MBTC Memory Access

Subtest 103 checks the capability of the board to access all allowed pages in main memory via the DMA and associated buffers.

### Subtest 104, MBTC Command Loopback

Subtest 104 checks the capability of the board to properly register command status for the executing and pending states. This subtest does not execute when **n** (the default) is entered selected for the test parameter prompt `Allow Manual Intervention`.

### Subtest 105, MBTC Chain Command Loopback

Subtest 105 checks the capability of the board to properly register command status while executing chaining commands. This subtest does not execute when **n** (the default) is entered for the test parameter prompt `Allow Manual Intervention`.

## Class 2 Subtests

Class 2 subtests verify operator controls, tape write protect capability, and unload reel capability. All subtests in this class are skipped if **n** (the default) is entered for the test parameter prompt `Allow Manual Intervention`.

**Table dev4200–5, Class 2 Subtests**

| SUBTEST | DESCRIPTION | TIME (min:sec) |
|---|---|---|
| 200 | MBTC Drive Online | Manual intervention time |
| 201 | MBTC Drive Write Protect | Manual intervention time |
| 202 | MBTC Drive Unload | Manual intervention time |

### Subtest 200, MBTC Drive Online

Subtest 200 verifies that drive online status is properly reflected to the software as the operator manipulates the drive controls. This subtest does not execute when **n** (the default) is entered for

the test parameter prompt `Allow Manual Intervention`.

### Subtest 201, MBTC Drive Write Protect

Subtest 201 verifies that file protection is properly reflected to the software as the operator removes and reinserts the write ring. This subtest does not execute when **n** (the default) is entered for the test parameter prompt `Allow Manual Intervention`.

### Subtest 202, MBTC Drive Unload

Subtest 202 tests the drive unload operation. This subtest does not execute when **n** (the default) is entered for the test parameter prompt `Allow Manual Intervention`.

## Class 3, 4, and 5 Subtests

Classes 3, 4, and 5 contain subtests to verify correct operation of the controller and drive for a wide selection of commands and data transfer scenarios, including chaining mode. These subtest are in the range of 300-599 (n00-n99). Subtest numbers in these three classes contain a prefix of *n* (as the hundreds digit), which is used to select tape density. The 300 series subtests are for 800 bpi tapes, 400 series for 1600 bpi, and 500 series for 6250 bpi.

**Table dev4200–6, Class 3, 4, and 5 Subtests**

| SUBTEST | DESCRIPTION |
|---|---|
| n00[1] | MBTC Write Tape Mark |
| n01 | MBTC File Skip Forward |
| n02 | MBTC File Skip Backward |
| n03 | MBTC Read File Marks Forward |
| n04 | MBTC Read File Marks Backward |
| n05 | MBTC Block Skip File Marks Forward |
| n06 | MBTC Block Skip File Marks Backward |
| n10 | MBTC Write Forward |
| n11 | MBTC Read Forward |
| n12 | MBTC Read Backward |
| n13 | MBTC Block Skip Forward |
| n14 | MBTC Block Skip Backward |
| n20[1] | MBTC Erase Gap |
| n25 | MBTC End-of-tape Sensing |
| n30 | MBTC Forced Parity Error |
| n50 | MBTC Write Files with Fixed Length Records |
| n51 | MBTC Read Files with Fixed Length Records |
| n52 | MBTC Write Files with Variable Length Records |
| n53 | MBTC Read Files with Variable Length Records |
| n60 | MBTC Write Chained Mode Data Files |
| n61 | MBTC Read Chained Mode Data File |

[1] *n* varies from 3 to 5; 3 is 800 bpi, 4 is 1600 bpi, and 5 is 6250 bpi.

The following execution times, which are for an STC 2920 drive at 1600 BPI, are approximate.

Other drives and densities will affect these figures. Specifying a block size larger than the default changes the runtimes for these subtests. All times except *n*00 include rewind time from the tape position at the end of the last subtest. If subtests *n*50-*n*53 are run to the end of the tape, then subtest times are appropriately lengthened, depending on the length of the tape.

**Table dev4200–7, Execution Times for $n = 4$**

| SUBTEST | TIMES (min/sec) |
|---|---|
| 400[1] | 00:10 |
| 401 | 00:15 |
| 402 | 00:20 |
| 403 | 00:10 |
| 404 | 00:20 |
| 405 | 00:10 |
| 406 | 00:20 |
| 410 | 00:10 |
| 411 | 00:15 |
| 412 | 00:20 |
| 413 | 00:10 |
| 414 | 00:15 |
| 420 | 00:10 |
| 425 | time to end of tape |
| 430 | 01:15 |
| 450 | 01:15 |
| 451 | 02:00 |
| 452 | 02:15 |
| 453 | 03:20 |
| 460 | 02:55 |
| 461 | 02:45 |

[1] Execution times for the 300 and 500 series subtests vary from those shown for the 400 series.

### Subtest n00, MBTC Write Tape Mark

Subtest *n*00 rewinds the tape and then writes a pattern of 100 tape marks. Media errors are ignored unless the count exceeds 5.

> **NOTE**
>
> In this subtest and all the following subtests, *n* varies from 3 to 5 based on tape density; 3 is 800 bpi, 4 is 1600 bpi, and 5 is 6250 bpi.

### Subtest n01, MBTC File Skip Forward

Subtest *n*01 rewinds the tape and then uses the file *skip forward* function to skip 90 tape marks written by Subtest *n*00. The subtest ignores media errors unless the count exceeds 10.

### Subtest n02, MBTC File Skip Backward

Subtest *n*02 rewinds the tape, skips 90 tape marks written by Subtest *n*00 in the forward direction, and then skips 80 tape marks in the reverse direction. This subtest ignores media errors unless the count exceeds 10 for tape motion in either direction.

### Subtest n03, MBTC Read File Marks Forward

Subtest *n*03 rewinds the tape, and then reads 90 tape marks written by Subtest *n*00 with a *block read* function in the forward direction. Media errors are ignored unless the count exceeds 10.

### Subtest n04, MBTC Read File Marks Backward

Subtest *n*04 rewinds the tape, uses the *file skip forward* function to skip 90 tape marks written by Subtest *n*00, and reads 80 tape marks backward, using the *block read backward* function. This subtest ignores media errors unless the count exceeds 10 in either direction.

### Subtest n05, MBTC Block Skip File Marks Forward

Subtest *n*05 rewinds the tape and then uses the *block skip forward* function to skip over 90 tape marks written by Subtest *n*00. The subtest ignores media errors unless the count exceeds 10.

### Subtest n05, MBTC Block Skip File Marks Backward

Subtest *n*06 rewinds the tape, uses the *file skip forward* function to skip 90 tape marks written by Subtest *n*00, and then it skips 80 tape marks backward using the *block skip backward* function. The subtest ignores media errors unless the count exceeds 10 in either direction.

### Subtest n10, MBTC Write Forward

Subtest *n*10 rewinds the tape and then writes a data pattern composed of 200 records, each containing incrementing byte values, modulo 256, beginning with zero, followed by 5 tape marks. The number of bytes entered at the test parameter prompt `Fixed Length Block Size` determines the record size; the default record size is 512 bytes. Media errors and block size errors are ignored unless the total count exceeds 10. Errors writing the tape marks are ignored if at least one is valid.

### Subtest n11, MBTC Read Forward

Subtest *n*11 first rewinds the tape and then reads forward 190 records written by Subtest *n*10. This subtest ignores media, block size, and data errors unless the total error count exceeds 10.

### Subtest n12, MBTC Read Backward

Subtest *n*12 rewinds the tape, skips the pattern written by Subtest *n*10 using the *file skip* function, and then the pattern is read backward for 190 records. The subtest ignores media, block

size, and data errors unless the total error count exceeds 10.

### Subtest n13, MBTC Block Skip Forward

Subtest *n*13 rewinds the tape and then skips over 190 tape blocks written by Subtest *n*10 using the *block skip* function. Media errors are ignored unless the error count exceeds 10.

### Subtest n14, MBTC Block Skip Backward

Subtest *n*14 rewinds the tape, skips the pattern written by Subtest *n*10 using the *file skip forward* function, and then it skips 190 tape blocks using the *block skip backward* function. Media errors are ignored unless the error count exceeds 10.

### Subtest n20, MBTC Erase Gap

Subtest *n*20 rewinds the tape, performs at least 50 erase-gap functions, and then performs a *block skip backward* function. The *block skip backward* function is timed to assure that the long gap exists. The subtest ignores media errors when erasing unless the count exceeds 5.

### Subtest n25, MBTC End-of-Tape Sensing

Subtest *n*25 rewinds the tape and performs a *write forward* operation until the EOT marker is read. This subtest executes only when **y** to the `Run EOT Subtest n25` test parameter prompt.

### Subtest n30, MBTC Force Parity Error

Subtest *n*30 rewinds the tape and writes a data pattern with the controller set to force parity errors. The forced errors are verified. The subtest ignores media and block size errors unless the total count exceeds 10.

Then the subtest rewinds the tape and a valid data pattern is written without forced errors. Media and block size errors are ignored unless the total count exceeds 10. Then the subtest rewinds the tape again and reads with forced parity errors. The subtest ignores media, block size, and data errors unless the total count exceeds 10.

### Subtest n50, MBTC Write Files with Fixed Length Records

Subtest *n*50 rewinds the tape, and then writes 10 files of data patterns. Each file consists of 200 fixed-length data blocks and is terminated with a tape mark. A second tape mark follows the last file. Each data block, which has its length determined by the number entered for the test parameter prompt `Fixed Length Block Size` (default 512 bytes), consists of increasing byte values, modulo 256, beginning with the block number (modulo 256) in the file. The first byte in each block is an exception; it contains the alternating values 0xaa and 0x55. When the end of tape marker is read the first byte contains the value 0xff.

The subtest recovers media errors by using a backspace plus erase gap plus rewrite technique. Any media error is retried the number of times specified in the test parameter prompt `Media Error Retry Count` (the default is 5).

The end of tape causes orderly (but premature) termination of the written data structure compatible with Subtest *n*51. Entering `y` to the test parameter prompt `Run Subtests n50-n53 to EOT` causes the subtest to write as many files of data as are required to fill the tape.

The error statistics which are displayed at the end of the subtest, can also be displayed as the errors occur by entering `y` to the `Display Each Media Error` test parameter prompt.

### Subtest n51, MBTC Read Files with Fixed Length Records

Subtest *n*51 rewinds the tape and then reads and verifies the data structure written by Subtest *n*50. The subtest recovers media errors by using a backspace plus reread technique. Any media error is retried the number of times specified for the test parameter prompt `Media Error Retry Count` (default is 5). The subtest detects the end of tape (if any) by reading software marks written by Subtest*n*50.

The error statistics which are displayed at the end of the subtest, can also be displayed as the errors occur by entering `y` to the `Display Each Media Error` test parameter prompt.

### Subtest n52, MBTC Write Files with Variable Length Records

Subtest *n*52 rewinds the tape and then writes one file to the tape. The file is terminated with two tape marks. The data in the file consists of records of various lengths, from 18 to 128*1024-1 bytes. Each record contains increasing byte values, modulo 256, beginning with the block size. The first byte in each block is the end of tape marker, which normally has a value of 0xaa. The first byte has a value of zero after the end of tape marker is read.

The subtest recovers media errors by using a backspace plus erase gap plus rewrite technique. Any media error is retried the number of times specified in the test parameter prompt `Media Error Retry Count` (default is 5).

The end of tape causes orderly (but premature) termination of the written data structure compatible with the read Subtest *n*53. If `y` is entered to the test parameter prompt `Run Subtests n50-n53 to EOT`, the subtest writes as many files of data as are required to fill the tape.

The error statistics which are displayed at the end of the subtest, can also be displayed as the errors occur by entering `y` to the `Display Each Media Error` test parameter prompt.

### Subtest n53, MBTC Read Files with Variable Length Records

Subtest *n*53 rewinds the tape and then reads and verifies the data structure written by Subtest *n*52. The subtest recovers media errors by using a backspace plus reread technique. The subtest detects the end of tape by reading software marks (if any) written by Subtest *n*52.

The error statistics which are displayed at the end of the subtest, can also be displayed as the errors occur by entering `y` to the `Display Each Media Error` test parameter prompt.

### Subtest n60, MBTC Write Chained Mode Data Files

Subtest *n*60 rewinds the tape and then writes one file to the tape. The file consists of several records of various sizes written in chained mode. Two file marks follow the data. The subtest recovers media errors by using a backspace plus erase gap plus rewrite technique. Any media error are retried the number of times specified in the test parameter prompt `Media Error Retry Count` (default is 5).

The error statistics which are displayed at the end of the subtest, can also be displayed by entering `y` to the `Display Each Media Error` test parameter prompt.

### Subtest n61, MBTC Read Chained Mode Data File

Subtest *n*61 rewinds the tape and then reads and verifies the data structure written by Subtest *n*60. The subtest recovers media errors by using a backspace plus reread technique. Any media error are retried the number of times specified in the test parameter prompt `Media Error Retry Count` (default is 5).

The error statistics which are displayed at the end of the subtest, can also be displayed by entering `y` to the `Display Each Media Error` test parameter prompt.

# Error Messages

**Erase Gap Failed**
The gap erase function did not succeed in erasing a gap on the tape.

**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 *dev4200.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:

## Table dev4200–8, *setup_iop* Exception 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 or tape drive status dump, if applicable}
There is a problem in IOP processing. The following messages may appear:

## Table dev4200–9, IOP Processing Error Messages

| TEXT | MEANING |
|---|---|
| TIME OUT | no response to device command |
| CHAINING OVERRUN | program and tape drive got out of sync chaining |
| MEDIA ERROR | tape error |

**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.

**MBTC exec level command:** *"ccccccccc" (0xnn)*
This message gives a description of the command, where *ccccccccc* is the verbal description of the command.

**MBTC exec level command: expected** *"ccccccccc" (0xnn)* **actual** *"ccccccccc" (0xnn)*
This message gives the expected and actual command, where *ccccccccc* is the verbal description of the command.

**MBTC exec level unit:** *nn*
This message gives the unit number in the execution level of the controller.

**MBTC exec level unit: expected** *nn*, **actual** *nn*
This message indicates a discrepancy in the unit number in the execution level of the controller.

**MBTC cstat[1]** *0xnn* **means:**
{error message detail}
This indicates a controller error status byte at offset 0x18 of controller. (The error message detail appears in the next entry.)

**MBTC cstat[1]: expected** *0xnn*, **actual** *0xnn*, **differences:**
{error message detail}
This indicates a discrepancy in the controller status at offset 0x18. The interpretation may be:

```
LAST BYTE [IN]VALID
[NO] MULTIBUS DMA TIMEOUT
[NO] READ PARITY ERROR
[NO] TAPE MARK
PENDING COMMAND [NOT] ABORTED
TAPE ERROR SUMMARY BIT [NOT] SET
```

**MBTC cstat[2]** *0xnn* **means:**
{error message detail}
This gives the interpretation of the formatter status byte at controller offset 0x19. (Possible interpretations appear in the next message entry.)

**MBTC cstat[2]: expected** *0xnn*, **actual** *0xnn*, **differences:**
{error message detail}
This message indicates a discrepancy in the formatter status byte at offset 0x19. The error message detail may be:

```
FORMATTER COMMAND [IN]COMPLETE
FORMATTER COMMAND REJECTED
FORMATTER COMMAND ACCEPTED
[NO] DATA OVERRUN
[NO] DATA CHECK
[NO] FORMATTER PROM ERROR
[NO] FORMATTER CORRECTABLE ERROR
[NO] FORMATTER DATA BUS ERROR
```

**MBTC exec level density:** *ddddd (nn)*
This messages gives the name and number of the density setting.

**MBTC exec level density: expected** *ddddd (nn)*, **actual** *ddddd (nn)*
This message gives the names and numbers of the expected and actual density.

**MBTC dstat[2]** *0xnn* **means:**
{error message detail}
This gives the interpretation of the drive status byte at offset 0x22. (The interpretations appear in the next entry.)

**MBTC dstat[2]: expected *0xnn*, actual *0xnn*, differences:**
{error message detail}

This indicates a discrepancy in the drive status byte at offset 0x22. The message detail may be:

```
DIAG MODE LATCH [NOT] SET
[NO] UCE CONDITION
[NO] WRITE TAPE MARK CHECK
[NO] END DATA CHECK
[NOT] MTE CONDITION
[NO] VEL ERROR
```

**MBTC dstat[3] *0xnn* means:**
{error message detail}

This gives the interpretation of the drive status byte at offset 0x23. The interpretation may be:

```
[NO] CRC ERROR
```

**MBTC dstat[3]: expected *0xnn*, actual *0xnn*, differences:**
{error message detail}

This indicates a discrepancy in the drive status byte at offset 0x23. The interpretation may be:

```
[NO] CRC ERROR
```

**MBTC dstat[6] *0xnn* means:**
{error message detail}

This gives the interpretation of the drive status byte at offset 0x26. (The interpretations appear in the next entry.)

**MBTC dstat[6]: expected *0xnn*, actual *0xnn*, differences:**
{error message detail}

This indicates a discrepancy in the drive status byte at offset 0x26. The interpretation may be:

```
DRIVE [NOT] READY
DRIVE [NOT] ONLINE
FILE [NOT] PROTECTED
[NOT] AT BEGINNING OF TAPE
[NOT] BACKWARD MOTION
[NOT] HIGH DENSITY
[NOT] PAST END OF TAPE
```

**MBTC dstat[7] *0xnn* means:**
{error message detail}

This gives the interpretation of the drive status byte at offset 0x27. The message may be:

```
[NOT] WRITE FUNCTION
```

**MBTC dstat[7]: expected *0xnn*, actual *0xnn*, differences:**
{error message detail}
This indicates a discrepancy in the drive status byte at offset 0x27. The message detail may be:

```
[NOT] WRITE FUNCTION
```

**MBTC status *0xnn* means:**
{error message detail}
This gives the interpretation of the controller status byte at offset 0x10. The message detail appears in the next entry.

**MBTC status: expected *0xnn*, actual *0xnn*, differences:**
{error message detail}
This indicates a discrepancy in the controller status byte at offset 0x10. The message detail may be:

```
BLOCK MODE
BUFFER A ACTIVE
BUFFER B ACTIVE
CHAIN MODE
COMMAND [NOT] EXECUTING
COMMAND [NOT] PENDING
DIAGNOSTICS [NOT] ENABLED
FORCE PARITY ERROR
INTERRUPT [NOT] ENABLED
[NO] INTERRUPT REQUEST
USE NORMAL PARITY
```

**Pattern error at offset *0xnnnnnn* in buffer: expected *0xnn* actual *oxnn [0xnn 0xnn ...]***
This message indicates an error in data has been detected. The context following the erroneous byte may also be printed.

**Retry count for this record: *nn***
This message gives the number of retries required to overcome a media error. This message is generally appended to other messages.

**Tape block size error: Expected size: *nn*, actual size: *nn***
This message indicates an erroneous tape block size.

# dev4300

# Multibus Terminal Controller Test

## Overview

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

## Prerequisites and Required Equipment

In order for this test to run, a Systech controller must be operational. The following table lists the required hardware depending on the type of machine under test.

**Table dev4300–1, Hardware Requirements**

| C1, C120 | C200 Series |
|---|---|
| MCU | Memory System[1] |
| MAU | CPX |
| SPU | SP2 |
| IOP | IOP |
| MBCU | MBCU |
| | PIA |

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

## Test Invocation

The *dev4300* test executes under the Diagnostic Shell (*dshell*) and supports all the features of the *dshell*. The *dshell* permits tests to be initiated in any order.

To invoke the *dev4300* test, use the procedure shown in the following figure. All responses in **boldface** are entered by the user. The prompts and responses appear sequentially on the screen, one line at a time. All prompts and responses are shown in one figure for convenience.

## Figure dev4300–1, Test Invocation Sequence

```
(spu)> cd /mnt/test RETURN
(spu)> sysreset RETURN
(spu)> mminit -s RETURN
(spu)> dshell RETURN
: test dev4300 [-c [class number(s)]] [-s [subtest number(s)]] [+>filename] RETURN
```

**NOTE**

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

Entering only **test dev4300** executes all *dev4300* subtests sequentially. Execute a specific class(es) of subtest(s) or one or more individual subtests by using the **-c** or **-s** options, respectively. Detailed information for using these options can be found in the "Dshell Overview" chapter of this manual. The [+>*filename*] option allows the test results to be appended to *filename*.

The following alternate test invocation procedure may be required in some cases.

**CAUTION**

The user response, **initall**, is typically required if the *initall* utility has not been run since the last powerup. 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 if 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 dev4300–2, Alternate Test Invocation Sequence

```
(spu)> cd /mnt/test RETURN
(spu)> initall RETURN
(spu)> dshell RETURN
: test dev4300 [-c [class number(s)]] [-s [subtest number(s)]] [+> filename] RETURN
```

## 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 dev4300–2, Getting Help During Test Parameter Entry

| Character | Description |
|---|---|
| ? | Displays this help menu |
| h | Provides help for a specific prompt |
| i | Displays the Iioconfig file |

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

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

## Figure dev4300–3, Test Parameter Menu

```
                    DEFINE TEST PARAMETERS
     []         Encloses allowed input ranges or values
     ()         Encloses the default value
     ^          Returns to the previous prompt
     :nn        Returns to the prompt # NN
     :          Returns to the first unsatisfied prompt
     :?         Reviews previous entries

       PERIPHERAL CONFIGURATION DATA
      CCU       Chassis  TYPE          CSR     Int
     -------    -------  -------       ---     ---

Device 0 = user defined configuration

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

    Controller Models

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

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

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

    * is in default mask

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

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

---

At any time during the test parameter sequence, several options are available as denoted at the top of the Test Parmeter 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

? — Request help for the current prompt (if available)

^ — Return to the previous prompt

## Prompt Explanations

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

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

This prompt selects a device for testing from the system configuration. If **0** (default) is entered, then user-defined prompts are displayed to determine the configuration to be tested; otherwise, if a device is selected, the prompts are not displayed.

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

Enter the CCU slot number for the desired IOP. Then additional prompts are displayed requesting hardware configuration information.

```
Multibus Chassis [0-3]                          (0) ->
```

Enter the number of the chassis to be tested. (The drive is attached to a controller which is in a Multibus chassis.)

```
Controller Offset in Multibus [0x0-0xfff]
                                              (0x20) ->
```

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

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

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

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

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

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

Enter the model number of the controller under test.

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

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

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

Enter the baud rate to be used to test controllers.

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

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

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

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

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

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

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

**Figure dev4300–4, Sample Test Parameter Summary**

```
                TEST PARAMETER SUMMARY

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

          Controller Models

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

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

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

         * is in default mask

Baud Rate Mask (hex)                                    : 0x523
Are test cables installed?  Enter ? for info            : y
Enter OK, or :NN to return to question NN              : OK
```

# 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.

# Class Descriptions

The *dev4300* test contains three classes of tests shown in the following table.

**Table dev4300–3, *dev4300* Test Classes**

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

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

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

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

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

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

Controller commands not formally tested include the following:

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

## Class 1 Subtests

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

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

**Table dev4300–4, Class 1 Subtests**

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

### Subtest 100, Reset and Self-test

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

### Subtest 101, Firmware Check

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

### Subtest 102, Timer

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

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

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

### Table dev4300–5, MTI Nominal Timer Periods

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

#### Subtest 103, Buffer Reconfiguration

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

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

### Table dev4300–6, Buffer Reconfigurations

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

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

## Class 2 Subtests

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

## Table dev4300–7, Class 2 Subtests

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

### Subtest 200, Port Configuration, Internal Loopback

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

#### Stop Bit Programming

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

#### Parity Bit Programming

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

#### Character Length Programming

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

#### Baud Rate Programming

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

### RTS Assertion Test

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

### USART Force Break and Framing Error Test

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

### DTR Assertion Test

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

### USART Transmitter Enable Test

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

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

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

## Subtest 201, Input Channel Configuration, Internal Loopback

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

## Subtest 202, Output Channel Configuration, Internal Loopback

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

## Subtest 203, Input Channel Termination Mask, Internal Loopback

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

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

### Subtest 204, Modem Configuration, Internal Loopback

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

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

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

### Subtest 206, Block Input, Internal Loopback

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

### Subtest 207, Block Output, Internal Loopback

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

## Class 3 Subtests

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

**Table dev4300–8, Class 3 Subtests**

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

### Subtest 300, Port Configuration, Physical Loopback

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

#### Stop Bit Programming

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

#### Parity Bit Programming

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

#### Character Length Programming

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

#### Baud Rate Programming

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

### Subtest 304, Modem Configuration, Physical Loopback

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

### Subtest 306, Block Input, Physical Loopback

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

### Subtest 307, Block Output, Physical Loopback

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

### Subtest 332, Command Error Code Verification, Physical Loopback

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

# Error Messages

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

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

**Command Error Code** *0xnn 0xnn*
<message text detailing command error>

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

## Table dev4300–9, Command Error Codes

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

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

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

**Erroneous Block I/O Termination Code, Port** *dd*
<message detailing termination code>

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

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

Expected and actual values of the termination cause are printed.

**Erroneous USART status code**
<message detailing status code>

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

**USART Status Code** *0xnn*
<message>

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

## Table dev4300–10, USART Error Messages

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

**Error allocating SPU window map: UNIX error message.**

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

**Error in load_iop**
<message detailing error>

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

**Error in recv_iop**
<message detailing error>

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

**Error in send_iop**
<message detailing error>

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

**Error in setup_iop**
<message detailing error>

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

**Table dev4300–11, IOP Access Error Messages**

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

**Error in start_iop**
<message detailing error>

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

**Error opening device**
<message detailing error>

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

**Error opening SPU window to main memory: UNIX error message.**

There is a system problem getting access to window map. Contact the Technical Assistance Center.

**Exception in mmalloc_int. Is main memory initialized?**

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

**Function status**: message
<possible port and channel indication>

There is a problem in IOP command processing. The following messages may appear:

**Table dev4300–12, IOP Command Processing Error Messages**

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

**Main memory allocation error**

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

**Port *dd* unpaired, not tested**

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

# dev4400

# Multibus Line Printer Test

## Overview

The *dev4400* test verifies that the Systech MLP-2000 line printer controller and the associated Centronics line printer are online and functional.

## Prerequisites and Required Equipment

For this diagnostic to run, a Systech MLP-2000 controller must be resident in a Multibus chassis associated with an IOP. A Systech MLP-2000 line printer controller and Centronics line printer must be operational. The Centronics line printer must be attached to either channel 0 or 1.

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

**Table dev4400–1, Hardware Requirements**

| C1, C120 | C200 Series |
|---|---|
| MCU | Memory System[1] |
| MAU | CPX |
| SPU | SP2 |
| IOP | IOP |
| MBCU | MBCU |
| | PIA |

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

## Test Invocation

The *dev4400* 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 *dev4400* 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 dev4400–1, Test Invocation Sequence

```
(spu)> cd /mnt/test RETURN
(spu)> sysreset RETURN
(spu)> mminit -s RETURN
(spu)> dshell RETURN
: test dev4400 [-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 dev4400** executes all *dev4400* 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 powerup. 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 if 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 dev4400–2, Alternate Test Invocation Sequence

```
(spu)> cd /mnt/test RETURN
(spu)> initall RETURN
(spu)> dshell RETURN
: test dev4400 [-c [class number(s)]] [-s [subtest number(s)]] [+> filename] RETURN
```

## 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 dev4400–2, Getting Help During Test Parameter Entry

| Character | Description |
|---|---|
| ? | Displays this help menu |
| h | Provides help for a specific prompt |
| i | Displays the */ioconfig* file |

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

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

## Figure dev4400–3, 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

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

Device 0 = user defined configuration

 1: Device Selection [0]                                (0) ->
 2: IOP [3-7][1]                                        (3) ->
 3: Multibus Chassis [0-3]                              (0) ->
 4: Controller Offset in Multibus [0x0-0xfff]
                                                    (0x8c0) ->
 5: Interrupt number [0-7]                              (3) ->
 6: Channel Number [0,1]                                (0) ->
 7: Print Interspace Time (sec) [0-1000]                (0) ->
 8: 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 Parmeter 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

**?** — Request help for the current prompt (if available)

**^** — Return to the previous prompt

## Prompt Explanations

Each prompt is repeated and explained in the following section.

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

This prompt selects a device for testing from the system configuration file (*/ioconfig*). If **0** (default) is entered, then user-defined configuration prompts (2-5 in the previous figure) are displayed to determine the configuration to be tested. If a specific device is to be selected, the number of the device is entered (**1**, **2**, etc...). At that point, the additional configuration prompts (2-5 in the previous figure) are not displayed.

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

Enter the CCU slot number for the desired IOP. Then additional prompts are displayed requesting hardware configuration information.

```
Multibus Chassis [0-3]                       (0) ->
```

Enter the number of the chassis to be tested. (The drive is attached to a controller which is in a Multibus chassis.)

```
Controller Offset in Multibus [0x0-0xfff]
                                          (0x8c0) ->
```

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

```
Interrupt number [0-7]                       (3) ->
```

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

```
Channel Number [0,1]                         (0) ->
```

Enter **0** if the printer is connected to channel 0 or enter **1** if the printer is connected to channel 1.

```
Print Interspace Time (sec) [0-1000]         (0) ->
```

Enter the amount of time (in seconds) to wait between sending patterns to be printed to the printer.

```
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 dev4400–4, Sample Test Parameter Summary**

```
                TEST PARAMETER SUMMARY
Device Selection                                : 0
IOP                                             : 3
Multibus Chassis                                : 0
Controller Offset in Multibus                   : 0x8c0
Interrupt number                                : 3
Channel Number                                  : 0
Print Interspace Time (sec)                     : 0
Enter OK, or :NN to return to question NN       : OK
```

# 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.

# Class Descriptions

The *dev4400* test contains the following class of subtests as shown in the following table:

**Table dev4400–3, *dev4400* Test Class**

| CLASS | DESCRIPTION |
|---|---|
| 2 | Pattern print routines |

## Class 2 Subtests

Class 2 subtests print a rotating character print pattern to the selected line printer. Class 2 subtests are listed in the following table:

**Table dev4400–4, Class 2 Subtests**

| SUBTEST | DESCRIPTION | TIME (min:sec) |
|---|---|---|
| 200 | Rapid Print Pattern | 1:00 |
| 201 | Slow Print Pattern | <:02 |

### Subtest 200, Rapid Print Pattern

Subtest 200 prints 150 consecutive lines of characters. The character pattern is rotated for each line so that each character is tested in each position on the line printer.

### Subtest 201, Slow Print Pattern

Subtest 201 prints one line of characters. If subtest looping is enabled, the subtest prompts for a delay between subtest executions which allows for printer adjustment.

# Error Messages

At several points in error processing a status dump may take place. A dump may have a status report from the following list:

**Ok**
No error.

**IOP bus error**
IOP bus error. Is the Multibus connected to the IOP? Is the controller configured for the correct address? Has the proper controller address been input as a test parameter?

**IOP address error**
IOP address exception. Should not occur.

**IOP cache error**
IOP cache error. IOP or controller hardware.

**IOP PBUS error**
IOP PBUS error. IOP hardware.

**Time_out**
Operation did not complete properly. May indicate hardware problems with controller or device.

**Controller busy**
Controller hardware error. Will not return to idle state.

**Premature interrupt**
Controller hardware error. Premature interrupt.

**Inv cmd to IOP**
Software error.

**Cntlr constant interrupt**
Controller hardware error. Constant interrupt.

**Controller error**
Unexpected controller status.

**Multiple interrupts**
Controller hardware problem.

**Memory alloc error**
Internal error.

**SPU/IOP req/ack overrun**
SPU/IOP intercommunication protocol error.

**SPU/IOP PBUS interrupt not acknowledged**
SPU/IOP intercommunication error. May be SPU or IOP hardware.

**Multibus error**
Multibus error interrupt. May be MBCU or controller problem.

**User termination**
IOP program terminated test. Usually a software problem.

**Boot_iop fork error**
Internal error.

**Boot_iop exit error**
Boot IOP problem.

**Call error**
Software error.

**Mmio error**
Main memory error. Assure main memory has been initialized.

**Error from ipcs_init**
Software problem.

**Timeout**
IOP unresponsive. May be IOP hardware problem.

**Hard error**
IOP crash. Reason should appear on test output before this error.

In addition, the following error messages may be displayed:

**IOP # error: undefined error 9.**
This should not occur. If it does, there is a problem in the SPU-Memory-IOP interface.

**Line printer 0/1 is offline, not ready, or disconnected.**
Line printer is offline, paper is not ready, controller is incorrectly addressed, incorrect channel was selected or incorrect controller was selected.

**IOP load failed.**
IOP could not be loaded with executable code. Assure that IOP load image resides in current directory or in */mnt/test*.

**SPU: Could not find main memory.**
Probable cause: main memory not initialized and/or reset.
Fix: run **sysreset** and/or **mminit**.

THIS PAGE INTENTIONALLY LEFT BLANK

# dev4410

# Multibus Plotter Test

## Overview

The *dev4410* test checks the IKON Versatec plotter controller, model 10085, and any attached Versatec plotter.

## Prerequisites and Required Equipment

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

**Table dev4410–1, Hardware Requirements**

| C1, C120 | C200 Series |
|---|---|
| MCU | Memory System[1] |
| MAU | MAU |
| 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).

If a plotter is not available, Class 1 subtests can be run as a check on the IKON controller only. If `y` is entered to the `Test Parameter Instructions` prompt in the test parameter menu, this procedure is described.

## Test Invocation

The *dev4410* 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 *dev4410* 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 dev4410–1, Test Invocation Sequence

```
(spu)> cd /mnt/test RETURN
(spu)> sysreset RETURN
(spu)> mminit -s RETURN
(spu)> dshell RETURN
: test dev4410 [-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 dev4410** executes all *dev4410* 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 powerup. 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 if 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 dev4410–2, Alternate Test Invocation Sequence

```
(spu)> cd /mnt/test RETURN
(spu)> initall RETURN
(spu)> dshell RETURN
: test dev4410 [-c [class number(s)]] [-s [subtest number(s)]] [+> filename] RETURN
```

## 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.

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 dev4410–3, 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

                    PERIPHERAL CONFIGURATION DATA
     CCU        Chassis Type    CSR     Int     Unit    Type
    -------     ------- ------- -----   ---     ----    --------
 1)  iop 3        0     PRC-002 0x2c0    1

Device 0 = user defined configuration

 1: Device Selection [0,1]                                    (0) ->
 2: IOP [3-7][1]                                              (5) ->
 3: Multibus Chassis [0-3]                                    (0) ->
 4: Controller Offset in Multibus [0x0-0xfff]              (0x2c0) ->
 5: Interrupt Number [0-7]                                    (1) ->
 6: Test Parameter Instructions [y,n]                         (n) ->
 7: Is IKON set for TTL interface [y,n]                       (n) ->
 8: Is IKON set for forced byte mode [y,n]                    (n) ->
 9: Suppress manual intervention [y,n]                        (n) ->

     Plotter models

     0: USER DEFINED                        12: 2000,2030   >64 char
     1:   80                                13: 2160
     2: 200A                 64 char        14: 8122
     3: 200A                >64 char        15: 8222
     4:  800                 64 char        16: 8124
     5:  800                >64 char        17: 8224
     6:  900                                18: 8136
     7: 1100                 64 char        19: 8236
     8: 1100                >64 char        20: 8124
     9: 1200                                21: 8242
    10: 1600                                22: 8172
    11: 2000,2030            64 char

10: Plotter model [0, 1-22]                                   (1) ->
11: Print characters per line [1-1000]                        (0) ->
12: SPP Print chars per line [1-1000]                         (0) ->
13: SPP scan lines for character [1-1000]                     (0) ->
14: Plot bytes per line [1-1000]                              (0) ->
15: Can this plotter print [y,n]                              (y) ->
16: Test ASCII print controls [y,n]                           (y) ->
17: Can this plotter plot [y,n]                               (n) ->
18: Does this plotter have shared print/plot mode
    [y,n]                                                     (y) ->
19: Shorten programmed output plot patterns [y,n]             (n) ->
20: Shorten DMA output plot patterns [y,n]                    (y) ->
21: Self test instructions [y,n]                              (n) ->
22: 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 Parmeter 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

? — Request help for the current prompt (if available)

^ — Return to the previous prompt

## Prompt Explanations

A description of the meaning of each prompt follows:

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

This prompt selects a device for testing from the system configuration. If the **0** (default) is entered, then user-defined prompts are displayed to determine the configuration to be tested; otherwise, if a device is selected, the prompts are not displayed.

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

Enter the CCU slot number for the desired IOP. Then additional prompts are displayed requesting hardware configuration information.

```
Multibus Chassis [0-3]                                    (0) ->
```

Enter the number of the chassis to be tested. (The drive is attached to a controller which is in a Multibus chassis.)

```
Controller Offset in Multibus [0x0-0xfff]                 (0x2c0) ->
```

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

```
Interrupt Number [0-7]                                    (1) ->
```

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

The following prompts display one line at a time after the configuration to be tested has been selected:

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

This prompt allows for the display of a short help file containing information about test strategy and the test parameters.

```
Is IKON set for TTL interface [y,n]                       (n) ->
```

Enter **y** to for TTL (Transistor-Transistor Logic) mode. Most Class 1 subtests (if selected) are skipped unless TTL mode is selected.

```
Is IKON set for forced byte mode [y,n]                    (n) ->
```

Enter either forced-byte mode or word-mode configuration for the IKON. Generally, word mode is selected.

```
Suppress manual intervention [y,n]                        (n) ->
```

If **y** is entered, Class 1 subtests, which require the plotter to be disconnected, are skipped. In addition, Class 3 subtests (plotter offline and out-of-paper subtests) are skipped. If **n** is entered, instructions such as:

```
Disconnect or reconnect plotter
Put plotter offline or online
Remove or restore plotter paper
```

are displayed when manual functions are performed.

---

```
Plotter model [0, 1-22]                                   (1) ->
```

If **0** (user-defined plotters) is entered, prompts are displayed for specifying the characteristics of the plotter being used. Entering a value in the range [1-22] causes the user-defined questions to be skipped.

The following four prompts are displayed for user-defined plotters.

```
Print characters per line [1-1000]                        (0) ->
```

Enter the number of characters to print per line.

```
SPP Print chars per line [1-1000]                         (0) ->
```

Enter the number of characters that the plotter can print in a single line when in shared print-plot mode (SPP).

```
SPP scan lines for character [1-1000]                     (0) ->
```

Enter the number of plot scans required to get the entire character block on the output medium.

```
Plot bytes per line [1-1000]                              (0) ->
```

Enter the number of bytes that can be plotted in a single-plot scan on the plotter under test.

```
Can this plotter print [y,n]                              (y) ->
```

Enter whether the plotter being tested has print capability.

```
Test ASCII print controls [y,n]                           (y) ->
```

This prompt is only displayed if **y** is entered to the previous prompt. If the printer has the capability to handle ASCII control characters, enter **y**.

```
Can this plotter plot [y,n]                               (n) ->
```

If the plotter being tested has plot capability, enter **y**; otherwise, press CTRL for the default.

```
Does this plotter have shared print/plot mode
[y,n]                                                    (y) ->
```

This prompt specifies whether the plotter has shared print-plot mode (print and plot can be overlaid).

```
Shorten programmed output plot patterns [y,n]            (n) ->
Shorten DMA output plot patterns [y,n]                   (y) ->
```

These prompts allow for shortening the plot-pattern generation. Normally, the full pattern is plotted in programmed output mode, and a short set is plotted in Direct Memory Access (DMA) mode.

```
Self test instructions [y,n]                             (n) ->
```

Enter **y** to display a help file giving instructions for operating the IKON self-test.

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

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

After entering all the parameters, the test parameter selections are echoed to the screen. If standard output is directed to a disk file, the test parameter summary is also directed to the disk file.

# 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.

# Class Descriptions

The *dev4410* test contains the following three classes of subtests as shown in the following table.

**Table dev4410–2, *dev4410* Test Classes**

| CLASS | DESCRIPTION |
|---|---|
| 1 | IKON 10085 controller and loopback tests |
| 2 | IKON/Versatec plotter status/interrupt/pattern tests |
| 3 | IKON/Versatec plotter offline/no-paper tests |

> **NOTE**
>
> The times shown are for the full subtest only (not the shortened version).

## Class 1 Subtests

Class 1 subtests verify some of the basic functionality of the controller. Class 1 subtests include board reset and loopback subtests.

Class 1 subtests are listed in the following table:

**Table dev4410–3, Class 1 Subtests**

| SUBTEST | DESCRIPTION | TIME (min:sec) |
|---|---|---|
| 100 | IKON Reset Capability | 1:40 |
| 101 | IKON Command Loopback | 1:40 |
| 102 | IKON Plotter Exception Loopback | :26 |
| 103 | IKON Port Selection | 1:40 |
| 104 | IKON Plotter Mode Selection | 1:40 |
| 105 | IKON Programmed Output Loopback | :11 |
| 106 | IKON DMA Output Loopback | 2:00 |
| 107 | IKON Data Output Interrupt Capability | :01 |

Standard ASCII printables are used for print patterns. Plotters may have special characters available for use that are normally considered ASCII control characters. Since some of these characters execute control functions that vary according to user specification, all tests avoid control characters, except for carriage return, newline, form feed, and end-of-transmission. If possible, use a plotter self-test to see all printable characters.

The following describes those items not formally tested:

- Versatec 8xxx series remote toner-on and toner-off commands
- Versatec 8xxx series status signals for low toner supply, toner-on/off status, and data transfer errors
- Versatec 8xxx series ASCII control characters (in print mode) which may be used to turn the toner subsystem on and off
- Serial interface plotters

- ASCII DC1 capability to switch to plot mode
- Option (Centronics) port on the IKON controller

### Subtest 100, IKON Reset Capability

Subtest 100 assures that the controller resets properly. This subtest checks the diagnostic command register and status register for the proper values. The controller must generate no interrupt. A plotter does need to be attached for this subtest.

### Subtest 101, IKON Command Loopback

Subtest 101 tests the controller command repertoire (remote buffer clear, end-of-line, form feed, end-of-transmission) using an internal loopback mechanism.

### Subtest 102, IKON Plotter Exception Loopback

Subtest 102 executes only when the associated test parameters specify this controller as being set for TTL mode and manual intervention is not suppressed. This subtest tests the offline and no-paper exceptions by forcing them through a loopback test mechanism provided for the TTL interface.

### Subtest 103, IKON Port Selection

Subtest 103 verifies Versatec versus Option (Centronics compatible) port selection by checking a status bit. The Option port is not tested. A plotter does not need to be attached for this subtest.

### Subtest 104, IKON Plotter Mode Selection

Subtest 104 checks that plotter mode selection is properly echoed in the status register. A plotter does not need to be attached for this subtest.

### Subtest 105, IKON Programmed Output Loopback

Subtest 105 executes only when the associated test parameters specify this controller as being set for TTL mode and manual intervention is not suppressed.

This subtest performs programmed output of byte patterns, which are checked via the loopback mechanism in the TTL interface. All possible byte patterns are tested by sequentially passing the 256 possible byte values. A random pattern of `0x10001` bytes is tested, using *rand()* and *srand()* with a seed value of 137.

### Subtest 106, IKON DMA Output Loopback

Subtest 106 executes only when the associated test parameters specify this controller as being set for TTL mode and manual intervention is not suppressed.

This subtest performs Direct Memory Access (DMA) output of byte patterns, which are checked via the loopback mechanism in the TTL interface. All possible byte patterns are tested by sequentially passing the 256 possible byte values. Many random patterns are tested by using *rand()* and *srand()* with a seed value of 137 plus the buffer size. This subtest tests the following DMA buffer sizes (hex values):

**Table dev4410–4, DMA Buffer Sizes**

| INITIAL SIZE | SIZE INCREMENT | SIZE LIMIT |
|---|---|---|
| 1 | 1 | 10 |
| 11 | 11 | 100 |
| 101 | 111 | ED0 |
| ED1 | 1 | EF0 |
| EF1 | 101 | 1000 |
| 1001 | 1001 | 10000 |
| FFFD | 1 | 10000 |

Precautions are taken to detect any improper byte swapping by a controller operating in auto-optimizing word mode.

### Subtest 107, IKON Data Output Interrupt Capability

Subtest 107 executes only if the associated test parameters specify this controller as being set for TTL mode and manual intervention is not suppressed. This subtest tests the IKON capability to generate an interrupt when output data circuits go idle.

## Class 2 Subtests

Class 2 subtests verify correct operation of the controller and plotter combination. Patterns are plotted to verify correct operation. Class 2 subtests are listed in the following table:

**Table dev4410–5, Class 2 Subtests**

| SUBTEST | DESCRIPTION | TIME (min:sec) |
|---|---|---|
| 200 | IKON/Versatec Status Reporting | :19 |
| 201 | IKON/Versatec Interrupt Reporting | :01 |
| 202 | IKON/Versatec FF/EOT Busy Check | 2:10 |
| 210 | IKON/Versatec Programmed Output Print | 4:40 |
| 211 | IKON/Versatec DMA Output Print | 4:40 |
| 220 | IKON/Versatec Programmed Output Plot | 4:30 |
| 221 | IKON/Versatec DMA Output Plot | 7:35 |
| 230 | IKON/Versatec Programmed Output Shared | 9:00 |
| 231 | IKON/Versatec DMA Output Shared | 12:00 |

### Subtest 200, IKON/Versatec Status Reporting

Subtest 200 verifies the IKON/Versatec capability to generate proper status for conditions "device ready," "device/interface ready," "data output complete." The subtest may plot some unlabeled data; it is to be ignored.

### Subtest 201, IKON/Versatec Interrupt Reporting

Subtest 201 verifies the IKON/Versatec capability to generate proper interrupt for conditions "device ready," "device/interface ready," "data output complete." This subtest also tests the capability to force "device ready" internally to the controller. The subtest may plot some unlabeled data; it is to be ignored.

### Subtest 202, IKON/Versatec FF/EOT Busy Check

Subtest 202 tests the capability of form feed and end-of-transmission to keep the plotter busy until all previously transmitted data is plotted. Visually verify the plotted output for correctness. Instructions for verification are plotted on the patterns.

### Subtest 210, IKON/Versatec Programmed Output Print

Subtest 210 executes only if the associated test parameter specifies that the plotter has print capability. All plotter controls are tested.

The subtest prints test patterns, including interpretation instructions, using programmed output. Visually check all patterns for errors. The test generates the following patterns:

- Rotating pattern of ASCII printable characters
- Sail pattern with X's, using remote line terminate function
- Truncated sail pattern, terminating lines with ASCII newline
- Truncated sail pattern, terminating lines with ASCII carriage return
- Truncated sail pattern, terminating lines with ASCII newline plus carriage return
- Single-line pattern testing remote buffer clear function
- Empty pattern testing remote reset function
- Simple pattern testing remote form-feed function
- Simple pattern testing ASCII form-feed capability
- Simple pattern testing remote end-of-transmission function
- Simple pattern testing ASCII end-of-transmission capability

Only the rotating pattern prints if the shortened programmed output test mode is selected with the associated test parameter. The ASCII control capabilities in the truncated sail patterns, the simple pattern testing ASCII form-feed capability, and the simple pattern end-of-transmission capability are not tested unless requested by the associated test parameter.

### Subtest 211, IKON/Versatec DMA Output Print

Subtest 211 executes only if the associated test parameter specifies that the plotter has print capability. The subtest checks all plotter controls.

The test patterns, described in Subtest 210, are printed, using DMA output with interrupt on "device/interface ready." Only the rotating pattern prints the shortened DMA test mode is selected with the associated test parameter.

### Subtest 220, IKON/Versatec Programmed Output Plot

Subtest 220 executes only if the associated test parameter specifies that the plotter has plot capability. The subtest checks all plotter controls.

This subtest plots test patterns, including instructions for interpretation, using programmed output. Visually check all patterns for errors. The subtest generates the following patterns:

- Nib-grid pattern (checks each plot nib so that failing nibs may be visually isolated)
- Sail pattern of small bars
- Simple pattern testing remote buffer clear function
- Simple pattern testing remote reset function
- Simple pattern testing remote form-feed function
- Simple pattern testing remote end-of-transmission function

The subtest generates only the nib-grid pattern if the shortened programmed output test mode is selected with the associated test parameter.

### Subtest 221, IKON/Versatec DMA Output Plot

Subtest 221 executes only if the associated test parameter specifies that the plotter has plot capability. The subtest checks all plotter controls.

This subtest plots the test patterns, described in Subtest 220, using DMA output with interrupt on "device/interface ready." Only the nib-grid pattern is generated if the shortened DMA test mode is selected with the associated test parameter.

### Subtest 230, IKON/Versatec Programmed Output Shared

Subtest 230 executes only if the associated test parameter specifies that the plotter uses shared print/plot mode. The subtest checks all applicable plotter controls.

The subtest patterns are printed/plotted using shared print/plot mode and programmed output. Visually check all patterns for errors; each pattern provides instructions for interpretation. The subtest generates the following patterns in shared plot mode (print buffer empty):

- Nib-grid pattern (checks each plot nib so that failing nibs may be visually isolated)

- Sail pattern of small bars
- Simple pattern testing remote buffer clear
- Simple pattern testing remote reset

The subtest generates the following patterns in shared print mode (using zero-filled plot buffers to scan out the print):

- Rotating pattern of printables
- Sail pattern of X
- Truncated sail pattern of X, using ASCII newline to terminate print buffers
- Truncated sail pattern of X, using ASCII carriage return to terminate print buffers
- Truncated sail pattern of X, using ASCII carriage return plus newline sequence to terminate print buffers
- Empty pattern to test remote buffer clear function
- Empty pattern to test remote reset function

Also, the subtest generates a "close shades on X" pattern, using actual data in both print and plot buffers.

The "truncated sail patterns" (ASCII) are not generated unless enabled by the associated test parameter. Also, only the "close shades on X" pattern is generated when the shortened programmed output test mode is selected with the associated test parameter.

**Subtest 231, IKON/Versatec DMA Output Shared**

Subtest 231 executes only if the associated test parameter specifies that the plotter uses shared print/plot mode. The test patterns, described in Subtest 230, are printed/plotted using shared print/plot mode and DMA output with interrupt on "device/interface ready." The subtest only generates "close shades of X" pattern when the shortened DMA test mode is selected.

## Class 3 Subtests

Class 3 subtests verify correct operation of the controller and plotter when offline and out-of-paper conditions exist. Class 3 subtests are listed in the following table:

**Table dev4410–6, Class 3 Subtests**

| SUBTEST | DESCRIPTION | TIME (min:sec) |
|---|---|---|
| 300 | IKON/Versatec Offline Status/Interrupt | N/A[1] |
| 301 | IKON/Versatec Out-of-Paper Status/Interrupt | N/A |

[1] Manual intervention time

### Subtest 300, IKON/Versatec Offline Status/Interrupt

Subtest 300 tests the capability to properly reflect plotter online status and the associated interrupt capability. This subtest does not execute if manual intervention is suppressed with the associated test parameter.

### Subtest 301, IKON/Versatec Out-of-paper Status/Interrupt

Subtest 301 tests the capability to properly reflect plotter out-of-paper status. The subtest checks online status, instead, for differential plotters. For TTL plotters, it tests the associated out-of-paper interrupt. This subtest does not execute if manual intervention is suppressed with the associated test parameter.

## Error Messages

**8259A status: poll | IRR | IMR | ISR** *0xnnnnnnnn*

The complete status of the 8259A interrupt controller is presented. The first byte is the poll status; the next, the interrupt request register; the third, the interrupt mask register; and the fourth, the in-service register.

**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 expected interrupt code** *0xnn*
{Interpretation}
**IKON actual interrupt code** *0xnn*
{Interpretation}

An interrupt of one type occurred but an interrupt of another type was expected. Refer to "Missing interrupt" 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 *dev4410.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:

**Table dev4410–7, 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 appear:

**Table dev4410–8, IOP Processing Error Messages**

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

**IKON diag dcmd 0xnn means:**
{interpretation of byte}

This logs a diagnostic command loopback byte. (The interpretation appears in the next entry.)

**IKON diag dcmd: expected** 0xnn, **actual** 0xnn, **differences:**
{interpretation of difference between expected and actual}

The interpretation may be one or more of the following, where the brackets enclose an optional *NOT* indicator:

**Figure dev4410–4, Interpreted Error Messages**

```
 UN:  UNUSED BITS SET
_VR:  [NOT] VERSATEC RESET
_VC:  [NOT] VERSATEC CLEAR
_VF:  [NOT] VERSATEC FORM FEED
_VT:  [NOT] VERSATEC END OF XMIT
_VL:  [NOT] VERSATEC END OF LINE
```

**IKON status** 0xnn **means:**
{interpretation}

This is the IKON status byte. The interpretation may be:

**Figure dev4410–5, Interpreted Status Byte Error Messages**

```
_OP:  VERSATEC PORT
_OP:  OPTION PORT

 DR:  DEV READY (diag)
 DR:  DEV NOT READY (diag)

 IR:  DEV RDY & OUTPUT IDLE
 IR:  DEV BSY | OUTPUT ACTV

 DP:  OUTPUT IDLE
 DP:  OUTPUT ACTIVE

 PL:  PRINT MODE
 PL:  PLOT MODE

 SP:  NOT SHR'D PRT/PLOT
 SP:  SHR'D PRT/PLOT MODE

 OL:  DEV ON LINE
 OL:  DEV OFF LINE

 NP:  DEV NO PAPER
 NP:  DEV HAS PAPER
```

**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 interpretations).

**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. The interpretation may be:

**Figure dev4410–6, Interpreted Interrupt Error Messages**

```
IR0:  DEVICE/INTERFACE READY
IR1:  OUTPUT IDLE
IR2:  DEVICE READY
IR3:  DEVICE OUT OF PAPER
IR4:  DEVICE OFF LINE
```

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

The count of spurious interrupts was not as expected. Spurious interrupts are those which are actually 'spurious' (IKON 8259A spurious interrupt) or are extra, but otherwise valid interrupts. (An IKON 8259A spurious interrupt is a condition where the original interrupt cause vanishes before processor interrupt service performs the interrupt acknowledge.)

**Unexpected interrupt**
IKON actual interrupt code *0xnn*
{interpretation}

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

THIS PAGE INTENTIONALLY LEFT BLANK

# dev4500

# Multibus Ethernet Controller Test

## Overview

The *dev4500* test verifies that the EXOS/201 Ethernet controller board is functional. The test verifies the IOP-to-controller interface and triggers the controller board's built-in self-test.

## Prerequisites and Required Equipment

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

**Table dev4500–1, Hardware Requirements**

| C1, C120 | C200 Series |
|---|---|
| MCU | Memory System[1] |
| MAU | CPX |
| SPU | SP2 |
| IOP | IOP |
| MBCU | MBCU |
| | PIA |

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

> **NOTE**
>
> Also, an operational *delni* module must be available for the Class 2 subtests. The *delni* module must be connected to the board under test via CONVEX cable 601-150001-200.

## Test Invocation

The *dev4500* 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 *dev4500* 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 dev4500–1, Test Invocation Sequence

```
(spu)> cd /mnt/test RETURN
(spu)> sysreset RETURN
(spu)> mminit -s RETURN
(spu)> dshell RETURN
: test dev4500 [-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 dev4500** executes all *dev4500* 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 powerup. 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 if 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 dev4500–2, Alternate Test Invocation Sequence

```
(spu)> cd /mnt/test RETURN
(spu)> initall RETURN
(spu)> dshell RETURN
: test dev4500 [-c [class number(s)]] [-s [subtest number(s)]] [+>filename] RETURN
```

## 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 dev4500–2, Getting Help During Test Parameter Entry

| Character | Description |
|---|---|
| ? | Displays this help menu |
| h | Provides help for a specific prompt |
| i | Displays the */ioconfig* file |

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

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

## Figure dev4500–3, 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

                PERIPHERAL CONFIGURATION DATA
      CCU       Chassis Type    CSR     Int     Unit    Type
     -------    ------- ------- -----   ---     ----    ------
 1)  iop  3       0     LAN-001 0x4c0    1       0      ex

Device 0 = user defined configuration

 1: Device Selection [0,1]                          (0) ->
 2: IOP [3-7][1]                                    (3) ->
 3: Multibus Chassis [0-3]                          (0) ->
 4: Controller Offset in Multibus [0x0-0xfff]     (0x0) ->
 5: Controller Interrupt [0-7]                      (0) ->
 6: Subtest loop count [10-1000]                  (100) ->
 7: Transceiver connected to active ethernet? [y,n]
                                                    (y) ->
 8: 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 Parmeter 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

**?** — Request help for the current prompt (if available)

**^** — Return to the previous prompt

## Prompt Explanations

A description of the meaning of each prompt follows:

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

This prompt selects a device for testing from the system configuration. If **0** (default) is entered, then user-defined prompts are displayed to determine the configuration to be tested; otherwise, if a device is selected, the prompts are not displayed.

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

Enter the CCU slot number for the desired IOP. Then additional prompts are displayed requesting hardware configuration information.

```
Multibus Chassis [0-3]                       (0) ->
```

Enter the number of the chassis to be tested. (The drive is attached to a controller which is in a Multibus chassis.)

```
Controller Offset in Multibus [0x0-0xfff] (0x0) ->
```

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

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

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

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

Enter the number of times each subtest is to be repeated.

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

If **y** is entered, Class 2 subtests are not executed since the subtests require an inactive Ethernet for proper operation.

```
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 dev4500–4, Sample Test Parameter Summary**

```
                TEST PARAMETER SUMMARY

Device Selection                                : 0
IOP                                             : 3
Multibus Chassis                                : 0
Controller Offset in Multibus                   : 0x0
Controller Interrupt                            : 0
Subtest loop count                              : 100
Transceiver connected to active ethernet?       : y
Enter OK, or :NN to return to question NN       : OK
```

# 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.

# Class Descriptions

The *dev4500* test contains the following three classes of subtests as shown in the following table:

**Table dev4500–3, *dev4500* Test Classes**

| CLASS | DESCRIPTION |
|---|---|
| 1 | EXOS/201 Ethernet controller access test |
| 2 | EXOS/201 Ethernet controller functional tests |
| 3 | EXOS/201 Ethernet controller online test |

## Class 1 Subtest

The Class 1 subtest verifies the basic interface from the IOP to the controller. The subtest ensures the controller responds to the IO address expected, and passes an internal self-test. The Class 1 subtest is listed in the following table:

**Table dev4500–4, Class 1 Subtest**

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

**Subtest 100, Controller to Host Access**

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

## Class 2 Subtests

Class 2 subtests verify the ability of the controller to transmit and receive various Ethernet packets. While in the *Ethernet* mode (with the use of a *delni* module connected to the board under test), these subtests transmit packets using an internal "transmit with self-receive" command, and transmit messages to verify the controller's address filtering. The "scatter/gather" operation of the controller is also tested by targeting data blocks to specific memory blocks.

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

Each subtest in the 200 series resets and reconfigures the controller at the start of the subtest. This allows the subtests to be run in any order. Each subtest will loop for a user-entered number of passes. Class 2 subtests are listed in the following table:

**Table dev4500–5, Class 2 Subtests**

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

### Subtest 200, Real Physical Slot Xmit/Recv

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

### Subtest 201, Modified Physical Slot Xmit/Recv

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

### Subtest 202, Broadcast Slot Xmit/Recv

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

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

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

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

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

### Subtest 205, Foreign Address Rejection Xmit/Recv

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

### Subtest 206, Scatter/Gather Operation

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

## Class 3 Subtest

The Class 3 subtest can be executed on an active Ethernet which allows the test to execute without changing the configuration of the hardware. These subtests do not transmit or receive the *broadcast* type Ethernet packets. The Class 3 subtest is listed in the following table:

**Table dev4500–6, Class 3 Subtest**

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

**Subtest 300, Real Physical Slot Only Xmit/Recv**

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

# Error Messages

When the requested test has completed, a summary displays, using a standard format as shown in the following figure:

**Figure dev4500–5, Sample Pass/Fail Printout**

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

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

**Figure dev4500–6, Sample Error Message**

```
*****  Mon Mar  4 17:02:27 1985  *****
Test: dev4500.t  1.1    Class: 2    Subtest: 205 1.1   Count: 1  Error: 0
Failed:  Foreign address rejection xmit/recv test

Error: Unexpected message ID, ID: 0x4b810009, Cmd: 0xd
```

Then an *end message* is displayed in the format shown in the following figure:

## Figure dev4500–7, Sample End Message

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

Test 'dev4500.t' failed
Elapsed time:   0:02:17
:
```

# dev4510

# Multibus HYPERchannel Controller Test

## Overview

The *dev4510* test verifies the functionality of an IKON 10077-NSC Hyperchannel controller board. This board interfaces to the Network Systems Corporation (NSC) A400 Hyperchannel Adapter which, in turn, connects to the actual Hyperchannel bus.

## Prerequisites and Required Equipment

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

**Table dev4510–1, Hardware Requirements**

| C1, C120 | C200 Series |
|---|---|
| MCU | Memory System[1] |
| MAU | CPX |
| SPU | SP2 |
| IOP | IOP |
| MBCU | MBCU |
| | PIA |

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

**NOTE**

A loopback cable, CONVEX cable 601-330001-200, is required for Class 2 subtests.

**NOTE**

The controller board must be detached from the A400 during this test. The voltages must be within margin and the clock must be between 5Mhz and 11Mhz.

This test is *not* designed to verify the IOP or Multibus Control Unit (MBCU). However, the test is coded to protect itself against Multibus or IOP failures if such are encountered.

# Test Invocation

The *dev4510* 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 *dev4510* 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 dev4510–1, Test Invocation Sequence**

```
(spu)> cd /mnt/test RETURN
(spu)> sysreset RETURN
(spu)>-mminit -s RETURN
(spu)> dshell RETURN
: test dev4510 [-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 dev4510** executes all *dev4510* 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 powerup. 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 if 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 dev4510–2, Alternate Test Invocation Sequence**

```
(spu)> cd /mnt/test RETURN
(spu)> initall RETURN
(spu)> dshell RETURN
: test dev4510 [-c [class number(s)]] [-s [subtest number(s)]] [+> filename] RETURN
```

## 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 dev4510–2, Getting Help During Test Parameter Entry**

| CHARACTER | DESCRIPTION |
|---|---|
| ? | Provides the help information |
| h | Provides help for a specific prompt |
| i | Displays the */ioconfig* file |

After the information displays, the system beeps and redisplays the last prompt.

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

## Figure dev4510–3, 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
       ?    Prints an additional help menu

                PERIPHERAL CONFIGURATION DATA
        CCU     Chassis  TYPE     CSR    Int Unit Type
       ------- -------- ------- ------ --- ---- -------
1)     iop 3       0     LAN-002 0xf00  3   0   HYP-001

Enter device 0 to begin user-defined configurations

  1: Device Selection [0,1]                          (0) ->
  2: IOP [3-7][1]                                    (3) ->
  3: Multibus Chassis [0-3]                          (0) ->
  4: Controller Offset in Multibus [0x0-0xfff]
                                                  (0x5c0) ->
  5: Interrupt number [0-7]                          (5) ->
  6: 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 Parmeter 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

**?** — Request help for the current prompt (if available)

**^** — Return to the previous prompt

## Prompt Explanations

A description of the meaning of each prompt follows:

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

This prompt allows selection of a controller from the system configuration. If **0** (default) is entered, then user-defined prompts are displayed to determine the configuration to be tested. Otherwise, if a device is entered, the prompts are not displayed.

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

Enter the CCU slot number for the desired IOP. Then additional prompts are displayed requesting hardware configuration information.

```
Multibus Chassis [0-3]                       (0) ->
```

Enter the number of the chassis to be tested. (The drive is attached to a controller which is in a Multibus chassis.)

```
Controller Offset in Multibus [0x0-0xfff]
                                          (0x5c0) ->
```

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

```
Interrupt number [0-7]                       (5) ->
```

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

```
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 dev4510–4, Sample Test Parameter Summary**

```
            TEST PARAMETER SUMMARY

IOP                                          : 3
Multibus Chassis                             : 0
Controller Offset in Multibus                : 0x5c0
Interrupt number                             : 5
Enter OK, or :NN to return to question NN    : OK
```

# 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.

# Class Descriptions

The *dev4510* test contains the following two classes of subtests as shown in the following table:

**Table dev4510–3, *dev4510* Test Classes**

| CLASS | DESCRIPTION |
|---|---|
| 1 | IOP/IKON 10077-NSC communications tests |
| 2 | IKON 10077-NSC loopback tests |

Any subtest may be started by itself. There are no subtest dependencies. However, on an unknown board, it is best to execute all the subtests in their default order since the subtests increase in complexity.

## Class 1 Subtests

Class 1 subtests verify the basic interface from the IOP to the controller is functional. This ensures that the IOP can read and write the three control and status registers on the IKON board and reset the board. These tests can be run with or without the loopback switch on (switch 2 of block U26) and with or without the loopback cable. Therefore, the controller can be configured for loopback and then run both classes. Class 1 subtests are listed in the following table:

**Table dev4510–4, Class 1 Subtests**

| SUBTEST | DESCRIPTION | TIME (min:sec) |
|---|---|---|
| 100 | Master Clear and ISR/ICR Register Check | 0:01 |
| 101 | Master Clear Through PCR Register | 0:01 |

### Subtest 100, Master Clear and ISR/ICR Register Check

Subtest 100 checks basic communications between the IOP and Interface board using the Interface Status Register (ISR) and Interface Control Register (ICR). Basically, values are loaded into the ICR and read back in the ISR.

A master clear is performed through the ICR and verified since the pattern in the ISR after a master clear must be a binary 00X0XXX010000001 where X is an unknown state bit.

Four of the ISR bits can be programmed from the ICR without loopback. These four bits are altered with walking '1's and walking '0's patterns. This operation is repeated 1000 times.

### Subtest 101, Master Clear Through PCR Register

Subtest 101 checks that the Pulse Command Register (PCR) can successfully perform a master clear. First, a 4 bit pattern is written to the CYCL, FCN1, FCN2, and FCN3 bits of the ICR and then master clear is pulsed through the PCR. All four bits previously set high should now be low. The master clear is repeated 1000 times with a verify each time.

## Class 2 Subtests

Class 2 subtests are performed with the loopback switch of the IKON 10077-NSC board set on (switch 2 of block U26 on the board) and with a loopback cable attached to the board. These loopback features allow most of the board to be tested.

> **NOTE**
>
> A loopback cable, CONVEX cable 601-330001-200, is required for Class 2 subtests.

- Function and status bits in the control and status registers can be verified.
- 16-bit at a time transfers can be performed.
- DMA input and output of blocks of data can be tested.
- Interrupts from the IKON board to the A400 and to the IOP are exercised.

Class 2 subtests are listed in the following table:

**Table dev4510–5, Class 2 Subtests**

| SUBTEST | DESCRIPTION | TIME (min:sec) |
|---|---|---|
| 200 | Verify Programmability of ISR/ICR | 0:01 |
| 201 | Perform 16-bit Transfers | 0:02 |
| 202 | Perform DMA from Main Memory Using ICR | 3:50 |
| 203 | Perform DMA to Main Memory Using ICR | 2:05 |
| 204 | Perform DMA from Main Memory Using PCR | 1:45 |
| 205 | Perform DMA to Main Memory Using PCR | 1:00 |
| 206 | Verify DMA Transfers Without Interrupts | 0:10 |
| 207 | Verify IKON Responds to Attention Interrupt | 0:30 |

### Subtest 200, Verify Programmability of ISR/ICR

Subtest 200 performs a more comprehensive test of the ISR/ICR registers using the internal loopback enabled by switch 2 of switch block U26. This test is repeated 1000 times.

### Subtest 201 Perform 16-bit Transfers

Subtest 201 performs two-byte transfers with a rotating ones bit and then a rotating zero bit. This operation verifies that each communications line toggles and that there is no crosstalk on the lines. The pattern test is repeated 1000 times.

### Subtest 202, Perform DMA from Main Memory Using ICR

Subtest 202 verifies the controller can fetch bytes from main memory and write them to the output port. The last word transferred can be verified by reading the Input Data register (IDR). To ensure that every available window can be read, all windows are allocated and read from. Two bytes are read starting at each window address 0xXXX000 and 0xXXXffe. Then windows are set up for one large area of 64 Kbytes. DMA transfers of varying lengths of 1024, 2048, 4096, etc. bytes up to the 64k limit are performed. Each length of data is repeated for 32 data patterns consisting of walking '1's and walking '0's patterns. The ICR is used to start the DMA transfer.

### Subtest 203, Perform DMA to Main Memory Using ICR

Subtest 203 uses the controller's onboard DMA to transfer data to main memory from the Output Data Register (ODR). A two-byte pattern is stored in the ODR and then is written repeatedly to main memory until the entire DMA block has been transferred. DMA transfers vary in length from 1024, 2048, 4096, etc. bytes up to 64 Kbytes. Each length of data is repeated for 32 data patterns consisting of walking '1's and walking '0's patterns. The ICR is used to start the DMA transfer.

### Subtest 204, Perform DMA from Main Memory Using PCR

Subtest 204 performs the varying length DMA operation described in Subtest 202 but uses the PCR to start the DMA transfer.

### Subtest 205, Perform DMA to Main Memory Using PCR

Subtest 205 is the same as Subtest 203 but uses the PCR to start the DMA transfer.

### Subtest 206, Verify DMA Transfers Without Interrupts

Subtest 206 transfers varying lengths of data as in Subtest 202 and waits for the indication in the ISR that each transfer is performed. After the transfer completes, interrupts are also checked to verify none occurred. The DMA flag is then reset and verified that it reset properly. The next transfer then begins.

**Subtest 207, Verify IKON Responds to Attention Interrupt**

Subtest 207 performs a master clear using the ICR and confirms it was successful by inspecting the results in the ISR. Then it confirms no interrupt is pending and sets FCN2 (a special loopback bit) and IENB (interrupt enable) in the ICR. This operation is supposed to cause an interrupt as if the adapter had sent it. The interrupt is verified.

A second check verifies an ATTN interrupt occurs by inducing it with the loopback bit FCN2 and IENB (interrupt enable). The interrupt is verified.

# Troubleshooting Guide

This is a functional test. Standard error reporting routines will report whether a problem is internal or external to the controller board. In addition, many errors have associated with them a list of possible causes that are printed if long error reporting is selected within the dshell (long error reporting is the default). The IKON 10077-NSC Hyperchannel controller board is the only FRU covered by this test.

# Interprocessor Protocol

The IOP/SPU communication protocol is supplied by the Message-Based System (MBS).

# Error Messages

- **0x40 Sysreset of ccus and mem failed**

  The SPU UNIX command *sysreset ccus; sysreset mem* returned a non-zero value. Try the command manually. May require running *io4000* or *mem4000* to isolate the problem.

**0x42 IOP print utility memory error**

Error in memory access within MBS. IOP-generated output could not be printed.

**0x80 Message received on wrong subqueue**

Errors 0x80 through 0x83 should never occur. If they do, see the note at the end of this section.

**0x81 Wrong subqueue active and locked**

Refer to error 0x80.

**0x82 Wrong subqueue active and MBS error**

Refer to error 0x80.

**0x83 Wrong subqueue active, error invalid**

Refer to error 0x80.

**0x84 Driver executed for no apparent reason**

The IOP driver was brought into execution but no reason could be found. Normally, the driver is executed because of a message sent to it from the SPU or because a controller has interrupted the IOP with the correct interrupt signal. Refer to the note at the end of this section.

**0x85 Master Clear through the ICR failed**

When the master clear bit in the ICR was set, the ISR failed to return to the correct bit pattern. Probably a bad controller.

**0x86 Write/read of ISR bits failed**

Unable to alter the ISR bits. Probably a bad controller.

**0x87 Controller is not present**

The IOP received a bus error when attempting to write to the controller's registers. It is most likely the board is not inserted correctly in the Multibus chassis. The controller could be bad.

**0x88 Master Clear through the PCR failed**

When the master clear bit in the PCR was set, the ISR failed to return to the correct bit pattern. Probably a bad controller.

**0x89 Switch 2 of U26 on Cntlr is not ON**

An internal loopback was attempted which only succeeds if the described switch is set ON. An internal failure of the controller board is possible.

**0x8A Loopback cable is not installed**

An external loopback was attempted which only succeeds if the loopback cable is attached properly to both ports of the controller. The cable may be built incorrectly. Refer to the section Prerequisites and Required Hardware for the cable part number. An internal failure of the controller board is possible.

**0x8C DMA from memory failed compare**

The last two bytes transferred from memory are checked in the IDR for correctness. The compare failed. The failure could be located anywhere along the path from main memory to the controller board, the controller board itself, or the loopback cable.

**0x8D DMA to memory failed compare**

Refer to error **0x8C.**

**0x8E Expected interrupt did not occur**

The controller was expected to interrupt in less than 128 msec. No interrupt was detected in this time. Either the interrupt is strapped incorrectly on the board or the board is failing to interrupt when it should.

**0x8F Programmed I/O failed data compare**

The transmitted and received data from two-byte transfers failed. Either the loopback cable is not working properly, some component along the path from main memory to the controller is malfunctioning, or the controller itself is bad.

**0x92 IOP Window Allocation Failed**

Setup of a window from the IOP to main memory resulted in an error. Either the IOP or the driver software on the IOP failed. Refer to the note at the end of this section.

**0x93 ISR not reset when RDMA & RATN pulsed**

At the beginning of several tests, a master clear is performed by pulsing two bits in the ICR or PCR. This message is printed when the ISR is checked afterward and found not to be in a reset state.

**0x94 ISR not in proper state after DMA**

The ISR is checked after every DMA operation to verify attention is not set (ATTF = 0), DMA is active (DMAF = 1), and ready is true (REDY = 1). One of these conditions was not correct.

**0x95 Unexpected interrupt occurred**

An interrupt occurred when none was expected. This occurs in Subtests 206 and 207 during portions of the subtests when no interrupt is expected.

**0x96 Extended Write/read of ISR bits failed**

The test of bits in the ICR and ISR failed because the ISR did not correctly reflect changes made to the ICR.

**0xc0 Error when attempting to receive a message**

The MBS reported an error in attempting to provide the SPU with a message.

**0xc1 Timeout because queue locked**

The SPU received a message, but all ten attempts to read the message resulted in the error *MBS queue locked* being reported by the MBS.

**0xc2 Msg interrupt but no msg found**

The SPU attempted ten times to receive a message when an interrupt occurred, but all attempts failed with a *no message* error from the MBS.

**0xc4 Timeout awaiting msg**

SPU expected the IOP to send a message, but it did not receive a message. This error message indicates that the controller failed to generate an interrupt when done, or a break exists in the path of the interrupt signal, which prevents it from being detected.

**0xc5 Rtn msg from subqueue other than 7**

The SPU received a message to the wrong subqueue; all messages are expected to arrive on subqueue seven. Refer to the note at the end of this section.

**0xc6 MBS returned bad error code**

A call to an MBS routine returned an error code that was unrecognizable.

**0xc7 Block transfer routine failed**

A memory to memory transfer failed using the function 'blt'. Refer to the note at the end of this section.

**0xc8 Too many devices failed. Fail subtest**

This message indicates that the test is terminating because the device failure limit was exceeded.

**0xc9 Return msg not of valid type**

The SPU received a message, but it was not a device result or system error message. Refer to the note at the end of this section.

**0xca IOP's queue locked too long**

Ten attempts to send a message to the IOP failed, with a locked condition being reported by MBS.

**0xcb No room on SPU side for msg from the SPU to the IOP**

Ten attempts to send a message failed because no free message MBS blocks were available for the SPU to use.

**0xcc Return msg error**

An attempt was made to send a message from the IOP to the SPU but resulted in an MBS error being reported by the Message-Based System.

**0xcd Error while configuring I/O**

The IOP returned a configuration message that specified a controller that does not exist. Refer to the note at the end of this section.

**0xcf Too many controllers specified**

More than six controllers were specified during user inputs.

**0xd0 IOP echoed msg when none sent**

The SPU received an echo message, but no message had been sent. Refer to the note at the end of this section.

**0xd1 Bad cntlr # in returned msg**

A drive result message specified a bad controller number. Refer to the note at the end of this section.

**0xd2 Outstanding msg cnt greater than 0**

About to start a task in a subtest when a check uncovers that there are still messages expected back from the IOP for the last task. Refer to the note at the end of this section.

**0xd3 No msg err when sending to SPU**

Ten attempts to send a message from the IOP to the SPU resulted in a in use. Refer to the note at the end of this section.

**0xd4 MBS error when sending to SPU**

Same as error code 0xd3 except that MBS is reporting an 'MBS_ERROR'.

**0xd5 Invalid MBS error when sending to SPU**

MBS is failing. It reported back an error code which is invalid. Refer to the note at the end of this section.

**0xd6 MBS locked when sending to SPU**

Same as error code 0xd3 except that MBS is reporting an 'MBS_LCKD' error.

**0xd7 IOP driver device is bad**

A request to test an unknown device was sent from the SPU to the IOP. Immediately after the IOP was loaded, it was sent information about the device that was to be tested. However, the latest request to test the device had bad information in it about the device. Refer to the note at the end of this section.

**0xd8 MBS error during IOP msg recv**

Same as error code 0xd4 except that it occurred during receipt of a message from the SPU to the IOP.

**0xd9 MBS queue locked during receive**

Same as error code 0xd6 except that it occurred during receipt of a message from the SPU to the IOP.

**0xda MBS no message available**

Same as error code 0xd3 except that it occurred during receipt of a message from the SPU to the IOP.

**0xdb Data chk of echoed msg failed**

A data compare of all echoed messages occurs on the SPU. If the compare fails, MBS is having difficulty sending message between the IOP and SPU. A main memory or software error is indicated. Most likely main memory is not working properly so try *mminit* and then rerun. If it still fails, try *mem4000.* Refer to the note at the end of this section.

**0xdc Bad message type to get_rtn_msg**

A C-function called get_rtn_msg was called to receive an echoed message or the IOP did not properly flag a message so that it looked like an echoed message. On the SPU side, the routine mbs_wait should have been called if an echoed message is actually expected. Refer to the note at the end of this section.

**0xdd Processor queue setup failed**

A call to the OS system utility, *pqutil,* returned a non-zero value which indicates it failed. Try it manually at the SPU prompt by typing **pqutil -I**. If it fails for any reason, then it cannot initialize memory in the first eight pages of main memory. Try running *mem4000.* If that fails, try reloading */mnt/os* from a backup or release tape and try again. *Dev4100* and *dev4500* also use *pqutil* so maybe try them.

**0xde Open for window to main mem failed**

An open of the system file */dev/wndw* was attempted and failed. Verify */dev/wndw* exists and has proper permissions.

**0xe0 IOP load module (*.x00) not fnd**

A check was made of the current directory first and then */mnt/test* for a file called *dev4510.x00.* It was not found. It should exist in */mnt/test.* If it is not there, restore */mnt/test* from a backup or release tape.

**0xe1 Error attempting to load iop**

A system command to execute *loadccu* failed. First, run *sysreset* and *mminit* and then try it again. If this does not work, try typing **loadccu -3s /mnt/test/dev4510.x00** at the SPU prompt. If this fails, verify */mnt/os/loadccu* exists and has the right permissions. Also verify that */mnt/test/dev4510.x00* has the right permissions.

**0xe2 Main memory allocation error**

One large block of main memory needs to be allocated by the test on the SPU side but failed. Run *mminit* and then rerun the test. If it still fails, try *mem4000* to verify memory is functional.

**NOTES**

Certain error messages may indicate a software error. To verify for a software error, perform the following steps:

1. Perform *sysreset* and *mminit -s*.
2. Rerun the failing subtest. If the problem repeats continue with step 3. If no error occurs, record the error in the error log and look for recurrences of the error.
3. Reload */mnt/test* from a backup or release tape. Rerun the failing subtest. If the problem repeats, report it.

THIS PAGE INTENTIONALLY LEFT BLANK

# dev4600

# Multibus Emulator Controller Test

## 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 dev4600–1, Untested External Signals**

| DIRECTION | NAME |
|---|---|
| Output | READY H<br>GO H<br>END CYCLE H |

Also, the following external signals are only partially tested:

**Table dev4600–2, Partially Tested Signals**

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

## 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 dev4600–3, 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).

# 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 dev4600–1, Test Invocation Sequence**

```
(spu)> cd /mnt/test RETURN
(spu)> sysreset RETURN
(spu)> mminit -s RETURN
(spu)> 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 powerup. 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 if 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 dev4600–2, 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
```

## 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 dev4600–4, Getting Help During Test Parameter Entry

| Character | Description |
|---|---|
| ? | Displays this help menu |
| h | Provides help for a specific prompt |
| i | Displays the */ioconfig* file |

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

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

## Figure dev4600–3, 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 [0x0-0xfff]     () ->
 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 [0x0-0xfff]       () ->
 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 Parmeter 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

**?** — Request help for the current prompt (if available)

**^** — Return to the previous prompt

## 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 [0x0-0xfff]  () ->
```

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 [0x0-0xfff]    () ->
```

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 dev4600–4, 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.

# 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.

# Class Descriptions

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

**Table dev4600–5, *dev4600* Test Classes**

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

## 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 dev4600–6, 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 |

### 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.

### 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.

### Subtest 102, FCNx, STTx Status

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

### 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.

### Subtest 104, DMA Preparatory

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

### 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.

### 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.

### 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 0x*nnnn*, 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 0x*nnnn*.

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 dev4600–7, 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 |

### 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 0x*nnnn*, 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.

## 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 dev4600–8, Class 2 Subtest**

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

### 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).

## 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 dev4600–9, Class 3 Subtests**

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

### 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 dev4600–10, 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 |

### 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.

# 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 dev4600–11, 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 dev4600–12, 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 dev4600–5, 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.

# dev5130

# VMEbus SMD/ESDI Disk Test and Formatter

## Overview

Test *dev5130* is a multi-purpose disk test which allows the user to do the following:

- Verify Interphase 4200 VMEbus/Storage Module Device (V/SMD) and Interphase 4201 VMEbus/Enhanced Small Device Interface (V/ESDI) disk controllers and attached drives operate properly.
- Format or reformat up to 12 drives at one time.
- Verify previously formatted drives.
- Perform disk maintenance such as slipping of bad sectors and mapping bad tracks to alternate tracks. All maintenance operations can be performed nondestructively so user data can be retained.

**CAUTION**

Some subtests in *dev5130* are data destructive.

**WARNING**

CONVEX UNIX cannot be running concurrently with the *dev5130* test.

This test description is intended as a reference for users who have a good understanding of disk architecture and basic controller operations and want a tool that will allow them to track down and fix often difficult to locate problems with disks. For the casual user, in other words, the person who wants to format a disk now and then or slip a defective sector or verify a disk by reading all the usable parts, the user should refer to the section "Examples of Typical Usage of This Test" within this test description. It is suggested that the Table of Contents be used as an outline for finding information within this test.

The *dev5130* test is divided into two code bodies designated *dev5130.t* and *dev5130.x00*. The code bodies *dev5130.t* and *dev5130.x00* refer to Service Processor Unit (SPU) and VIOP resident code, respectively. The first body of code *dev5130.t* executes under the *dshell* from SPU UNIX. The primary function of this code body is to interface the user to the test and to control the testing sequence. This interface software consists of user prompts and error message printouts for all subtests except Subtest 300, Format Setup and Subtest 400, Interactive Test. The user prompts depend on the *dshell* extension routine *read_parm* for user inputs. The use of this routine allows the user interface of *dev5130.t* to be directly compatible with the other peripheral test programs.

Subtests 300 and 400 are interactive with their own prompts and set of commands including help commands.

The second body of code is *dev5130.x00* and executes under the Event Governed Operating System (EGOS). This body of code contains the command processing software used to test the controller(s) and SMD devices. The code communicates to the Service Processor via the Message-Based Subsystem (MBS) and through main memory.

# Prerequisites and Required Equipment

The SMD 4200 or ESDI 4201 controller must be configured for use. This is accomplished by changing jumpers on the board. Only changes to the VMEbus address of the controller are supported by this test. All other jumpers must be in a predefined configuration. All jumpers except the bus request block and address jumpers should be set properly by Interphase.

The configuration options and the CONVEX configuration are defined in the following table:

**Table dev5130–1, 4200 Controller Jumper Configurations**

| CONFIGURATION OPTION | POSSIBLE CONFIGURATIONS | CONVEX CONFIGURATIONS |
|---|---|---|
| Address Modifier | 0x29; 0x2D | 0x2D |
| Base Address | 0x0000,0x0200,...,0xFE00 | Any are possible |
| Bus Request Priority | 0-3 | 3 |

All other jumpers are for factory use only and should not be changed.

The test requires the system configuration in one of the next two table depending on the type of machine under test.

**Table dev5130–2, Hardware Requirements (C1, C120)**

| ITEM | MINIMUM | MAXIMUM |
|---|---|---|
| Interphase SMD 4200/ESDI 4201 controller | 1 | 12 |
| SMD device | 1 | 12 |
| Input/output processor (VIOP) | 1 | 5 |
| Multibus card cage | 1 | 10 (2/IOP) |
| Service processor unit (SPU) | 1 | 1 |
| VMEbus control unit (VBCU) | 1 | 12 |
| Memory control unit (MCU) | 1 | 1 |
| Memory array unit (MAU) | 4MB | System limit |

**Table dev5130–3, Hardware Requirements (C200 Series)**

| ITEM | MINIMUM | MAXIMUM |
|---|---|---|
| Interphase SMD 4200/ ESDI 4201 controller | 1 | 12 |
| SMD device | 1 | 12 |
| Input/output processor (VIOP) | 1 | 12 |
| Multibus card cage | 1 | 12 |
| Service processor unit (SP2) | 1 | 1 |
| VMEbus control unit (VBCU) | 1 | 12 |
| Memory system (one odd, one even) | 1 | System Limit |
| Peripheral interface adapter (PIA) | 1 | 4 |
| CPU utilities (CPX) | 1 | 1 |

# Test Invocation

Test *dev5130* executes under the Diagnostic Shell (Dshell), and supports most of the features of the Dshell. Control of failures is provided by the test since the Dshell does not support multiple drive operations where one drive could fail and yet the subtest should continue on the remaining drives. The Dshell permits the user to initiate subtests in any order. Therefore, each subtest is designed to be independent of all other subtests except in Class 3 which performs system formatting. The default order of subtest execution is from the least complex to the most complex operation.

To invoke the *dev5130* test, use the procedure shown in the following figure. All responses in **boldface** are entered by the user. The prompts and responses in the following figure would appear sequentially on the screen, one line at a time. All the prompts and responses are shown in one figure for convenience.

> **CAUTION**
>
> Entering only **test dev5130** should never be done on unformatted SMD/ESDI drives since Class 1 and Class 2 tests can overwrite the original Manufacturer's Defect map.

## Figure dev5130–1, Test Invocation Sequence

```
(spu)>  cd /mnt/test RETURN
(spu)>  sysreset RETURN
(spu)>  mminit -s RETURN
(spu)>  dshell RETURN

CONVEX DIAGNOSTIC SHELL

: test dev5130 [-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.

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 within this manual.

Entering only **test dev5130** executes all *dev5130* subtests sequentially except Class 4 which must be invoked with the **-c** or **-s** options. The [+> *filename*] option appends all test results to file *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 powerup. 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 if 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 dev5130–2, Alternate Test Invocation Sequence**

```
(spu)> cd /mnt/test RETURN
(spu)> initall RETURN
(spu)> dshell RETURN
: test dev5130 [-c [class number(s)]] [-s [subtest number(s)]] [+> filename] RETURN
```

## Test Parameter Menu

Once the test is invoked, the first prompt allows selection of defaults. If the test is run with defaults (by answering **y**, several prompts are skipped and the next prompt displayed is the `Device selection` prompt. If **n** is entered to the first prompt (no defaults are assumed), then a series of prompts are presented. The following figure shows all possible 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. To abort (terminate) the test, enter ^**c** (hold down the CTRL key and press **c**) during test execution. All possible 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 dev5130–4, Getting Help During Test Parameter Entry**

| Character | Description |
|---|---|
| ? | Displays this help menu |
| d | Displays *DB_diskfmt* file containing the drive parameters |
| e | Displays entered drive entries |
| h | Provides help for a specific prompt |
| i | Displays the */ioconfig* file |

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

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

## Figure dev5130–3, *dev5130* 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.
  ?         Prints an additional help menu
 1:  Default type (r[read only], w[rite allowed], n[o_defaults])
    [w,r,n]                                              (r) ->
 2:  Are writes to other than the diagnostics cylinder allowed
    [y,n]                                                (n) ->
 3:  Are writes to the diagnostics cylinder allowed
    [y,n]                                                (n) ->
 4:  Number of errors allowed per device each subtest
    [0-65535]                                            (0) ->
 5:  Number of devices failed after which test aborts
    [0-65535]                                           (12) ->
 6:  Threshold for new flaws found during pattern test
    [0-65535]                                            (0) ->
 7:  Verbosity of test [0-31]                            (3) ->
 8:  If pattern testing, start after last completed pattern
    [y,n]                                                (y) ->

            PERIPHERAL CONFIGURATION DATA
      CCU   Chassis  Type     CSR    Int Unit Type
    ------- ------- -------- ------ --- ---- --------
 1) viop 7    0     DKC-204  0x200  2   0   DKD-208
 2) viop 7    0     DKC-204  0xc00  6   0   DKD-214
Enter device 99 to begin user-defined configurations or -1 to end selection
 9:  Device selection [-1,1-2,99]                       (-1) ->
10:  Previously formatted on CONVEX system with Interphase 4200/4201 controller
     [y,n]                                               (n) ->
11:  Serial number (5 to 31 characters) []
                          (Default: read from drive) ->
12:  Device selection [-1,1-2,99]                       (-1) ->
Enter VIOP -1 to end user-defined configurations
13:  VIOP [-1,3-7]                                      (-1) ->
14:  VMEbus Chassis [0-1]                                (0) ->
15:  Controller Offset in VMEBus [0x0-0xffff]
                                                    (0xe00) ->
16:  Interrupt number [1-7]                              (2) ->
17:  Unit number [0-1]                                   (0) ->
18:  Drive name []                                 (DKD-208) ->
19:  Previously formatted on CONVEX system with Interphase 4200/4201 controller
     [y,n]                                               (n) ->
20:  Serial number (5 to 31 characters) []
                          (Default: read from drive) ->
Enter VIOP -1 to end user-defined configurations
21:  VIOP [-1,3-7]                                      (-1) ->
22:  Enter OK, or :NN to return to question NN [OK]
                                                        (OK) ->
```

At any time during the test parameter sequence, several options are available as denoted at the top of the Test Parmeter 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

**?** — Request help for the current prompt (if available)

**^** — Return to the previous prompt

## Prompt Explanations

A description of the meaning of each prompt follows:

---

```
Default type (r[read only], w[rite allowed], n[o_defaults])
[w,r,n]                                              (r) ->
```

> **CAUTION**
>
> If **w** is entered to this prompt, then destructive writes are allowed to the entire disk and loss of data and system information is possible.

If **w**, **r**, or RETURN is entered, several test parameter prompts for default selections are skipped and the default values are retained with the exception of the following two prompts:

```
 Are writes to other than the diagnostics cylinder allowed
[y,n]                                                (n) ->
 Are writes to the diagnostics cylinder allowed
[y,n]                                                (n) ->
```

The default values for these prompts are set to **y** if **w** is entered for the `Default type` prompt. They are set to **n** if **r** is entered. The configuration file is then displayed and drive selection begins. Enter **n** to change any of the default parameters.

The following prompts are displayed if the first prompt is answered with **n**:

```
 Are writes to other than the diagnostics cylinder allowed
[y,n]                                                (n) ->
```

If answered with **n** or RETURN, writes are not allowed to cylinders other than the diagnostic cylinder. However, if answered with **y**, writes are allowed to all cylinders with the exception of the diagnostic cylinder which is controlled by the next prompt.

```
 Are writes to the diagnostics cylinder allowed
[y,n]                                                (n) ->
```

If answered with **n** or RETURN, writes are not allowed to the diagnostic cylinder. However, if

answered with **y**, writes are allowed to the diagnostic cylinder.

```
Number of errors allowed per device each subtest
[0-65535]                                    (0) ->
```

If answered with **0** or (RETURN), a device fails if an error occurs during execution of the subtest. However, if answered with a value in the range [**1-65535**], a device fails if the number of errors that occur during the execution of the subtest exceeds this value.

```
Number of devices failed after which test aborts
[0-65535]                                   (12) ->
```

If a value in the range [**12-65535**] or (RETURN) is entered, the test will not terminate unless all devices fail because only 12 devices can be exercised at one time. However, if a value in the range [**0-11**] is entered, the test fails when the number of failed devices exceeds this limit.

```
Threshold for new flaws found during pattern test
[0-65535]                                    (0) ->
```

If **0** or (RETURN) is entered, a device fails if one or more new flaws are detected during pattern test. However, if a value in the range [**1-65535**] is entered, a device fails if the number of new flaws detected during pattern test exceeds this limit.

```
Verbosity of test [0-31]                     (3) ->
```

If **1** or (RETURN) is entered, a summary of all flaws for each drive is printed during Subtest 302, Fix Flaws. In general, verbosity is a sum of values which determines how informative the test is. The values which may be chosen are shown in the following table:

**Table dev5130–5, Verbosity Values and Printed Information**

| VALUE | PRINTED INFORMATION |
|---|---|
| 1 | Print summary of bad sectors during Subtest 302, Fix Flaws Subtest |
| 2 | Print each grown defect as it is found |
| 4 | Print Manufacturer's Defect and Grown Defect list reads |
| 8 | Print pattern during the interactive command *pattern_test* |
| 16 | Print all errors during the pattern test in Subtest 301, Format and Pattern Subtest |

For example, a value of 11 means print the data associated with values of 1, 2, and 8 (1+2+8=11).

```
If pattern testing, start after last completed pattern
[y,n]                                        (y) ->
```

If **y** or (RETURN) is entered, all drives begin pattern test after the last completed pattern. If the drives were previously using different pattern sets or any of the drives had not completed at least pattern 1, then pattern testing is restarted from the beginning on all drives. However, if **n** is entered, then pattern testing is restarted from pattern 1.

The following prompts are displayed if the first prompt is answered with **y**, or after the `If`

**pattern testing, start after last completed pattern prompt is answered:**

```
Device selection [-1,1-2,99]                    (-1) ->
```

The range [1-2] in the list of valid inputs corresponds to all the drives listed in the configuration file */ioconfig*. This range varies depending on how many drives are in the */ioconfig* file.

If answered with **-1** or (RETURN), the input is rejected since this is the first request for a device. If answered with one value in the range [**1-2**], then that drive is selected. If at least one drive has already been selected, the value entered must correspond to a different configuration file entry from the entry previously chosen. However, if answered with **99**, all configuration information in subsequent prompts must be manually entered.

The following prompts are displayed if a value in the range [**1-3**] is entered for the `Device Selection` prompt:

```
Previously formatted on CONVEX system with Interphase 4200/4201 controller
[y,n]                                           (n) ->
```

If **n** or (RETURN) is entered, then Class 1 and Class 2 diagnostics format the portions of the disk they need. In Class 3, if the drive has defect lists, then a message is displayed that the disk appears to be previously formatted and a prompt asks whether to continue this drive as a formatted drive. If **n** is entered, the drive is failed. However, if **y** is entered, then Class 1 and 2 diagnostics only perform formatting when the format command is being tested. In Class 3, if the drive does not appear to have defect maps, then a message is displayed that the disk appears to be unformatted and a prompt asks whether to continue this drive as an unformatted drive. If **n** is entered, the drive is failed. The Class 4 subtest ignores the input to this prompt.

```
Serial number (5 to 31 characters) []
                        (Default: read from drive) ->
```

If (RETURN) is entered, then **y** should have been entered to the previous prompt since the serial number must be read from a formatted drive. If **n** was entered to the previous prompt, a serial number is entered at this prompt. If a serial number is entered, it can be from 5 to 31 characters in length and can consist of lower or uppercase characters, numbers, and dashes. Internally, all letters are converted to uppercase so lower and uppercase inputs have the same result.

```
Device selection [-1,1-2,99]                    (-1) ->
```

The range [1-2] in the list of valid inputs corresponds to all the drives listed from the configuration file */ioconfig*. This range varies depending on how many drives are in the */ioconfig* file. If **-1** or (RETURN) is entered, the test exits device selection and asks the user to confirm that all inputs are ok. However, if **99** is entered, then manual inputs of configuration data begins. If one value in the range [**1-2**] is entered, then that drive is selected. If at least one drive has already been selected, the value must correspond to a different configuration file entry from the one previously chosen.

The following prompts are displayed if the `Device Selection` prompt is answered with **99**:

```
VIOP [-1,3-7]                                   (-1) ->
```

If **-1** or RETURN is entered and at least one drive has been chosen, then the user is asked to confirm that all inputs are okay. If no drives have been chosen, then this input (-1) is rejected. If the test is executing on a C1 or C120 machine and one value in the range **[3-7]** is entered, then this value selects the position of the VIOP in the Central Processing Unit (CPU) card cage in the C1 or C120. If the test is executing on a C200 Series machine and one value in the range **[0-3]** is entered, then this value selects the position of the VIOP in the CPU card cage in the C200 Series machine.

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

If **0** or RETURN is entered, then VMEbus chassis 0 is selected. There are two chassis numbered 0 and 1. If **1** is entered, the second chassis is selected.

```
Controller Offset in VMEbus [0x0-0xffff]
                                                    (0xe00) ->
```

If **0xe00** or RETURN is entered, then the controller which has been set to address 0xe00 is selected. This must be the controller to which the desired drive is attached. Offsets range from 0x0000 to 0xfe00 on 512 byte boundaries (for example, 0x0000, 0x0200, 0x0400, ... etc).

```
Interrupt number [1-7]                                  (2) ->
```

If **2** or RETURN is entered, then interrupt two is used by the controller when an I/O operation completes to interrupt the VIOP. However, if a value in the range **[1, 3-7]** is entered, then this interrupt is used. There is only one requirement when choosing interrupt numbers. Each controller in a VME chassis must have a unique interrupt number. Two controllers in different chassis can use the same interrupt number.

```
Unit number [0-1]                                       (0) ->
```

If **0** or RETURN is entered, then the drive must be set to unit address 0 (usually a switch setting internal to the drive). The cable with the small connector (B cable) from the drive must also be connected to the unit 0 socket on the controller. However, if **1** is entered, then the drive must be set to unit address 1 and the B cable from the drive must be connected to the unit 1 socket on the controller.

```
Drive name []                                     (DKD-208) ->
```

If **DKD-206** or RETURN is entered, then the drive must be an NEC D2352A 520-Mbyte drive. The current alternatives to this drive are the **DKD-208** which is the NEC D2363 1.1-GByte drive and the **DKD-214** which is the Hitachi DK514-38 380-Mbyte Removable Disk Storage (RDS) drive. The RDS drive has an ESDI interface so it must be connected to the Interphase 4201 ESDI controller while the other drives connect to the Interphase 4200 SMD controller.

Any SMD or ESDI drive that works with the Interphase controllers can be used. The drive must be added to the */mnt/bin/lib/DB_diskfmt* file.

```
Previously formatted on CONVEX system with Interphase 4200/4201 controller
[y,n]                                                   (n) ->
```

If the drive is unformatted, has been formatted on any other system than a CONVEX system, or has been formatted with a controller other than an Interphase 4200/4201 controller, then **n** is entered.

> **NOTE**
>
> Previously formatted means that the drive has completed Subtest 300, Format Setup Subtest, not that all of Class 3 subtests have been completed.

If **n** or (RETURN) is entered, then Class 1 and Class 2 diagnostics format the portions of the disk they need. In Class 3, if the drive has defect lists, then a message is displayed that the disk appears to be previously formatted and the user is asked whether to continue this drive as a formatted drive. If **n** is selected, the drive is failed. However, if **y** is entered, then Class 1 and Class 2 diagnostics only do formatting when the format command is being tested. In Class 3, if the drive does not appear to have defect maps, then a message is displayed that the disk appears to be unformatted and the user is asked whether to continue this drive as an unformatted drive. If **n** is entered, the drive is failed. The Class 4 subtest ignores the input to this prompt.

---

```
Serial number (5 to 31 characters) []
                    (Default: read from drive) ->
```

If (RETURN) is entered, then **y** should have been entered to the last prompt since the serial number must be either provided at this prompt or be read from the drive. If **n** is entered to the previous prompt, the serial number must be entered at this prompt. If a serial number is entered, it can be from 5 to 31 characters in length and can consist of lower or uppercase characters, numbers, and dashes. Internally, all letters are converted to uppercase so lower and uppercase inputs have the same result.

```
VIOP [-1,3-7]                                    (-1) ->
```

If **-1** or (RETURN) is entered and at least one drive has been chosen, then the user is asked to confirm that all inputs are okay. If no drives have been chosen, then this input (-1) is rejected. If the test is executing on a C1 or C120 machine and one value in the range [**3-7**] is entered, then this value selects the position of the VIOP in the Central Processing Unit (CPU) card cage in the C1 or C120. If the test is executing on a C200 Series machine and one value in the range [**0-3**] is entered, then this value selects the position of the VIOP in the CPU card cage in the C200 Series machine.

```
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.

```
*********************************************
* DATA DESTRUCTIVE OPERATION IS ABOUT TO START.*
* VERIFY WRITE-PROTECT IS SET ON ALL DRIVES    *
* EXCEPT THOSE YOU ARE TESTING/FORMATTING.     *
*********************************************
 Then reconfirm that you want to continue [yn] ->
```

If **y** is entered then the actual testing commences with the loading of a driver into the VIOPs and then the execution of subtests begins. However, if **n** is entered, the test terminates and returns to the SPU prompt.

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 dev5130–4, Sample Test Parameter Summary**

```
            TEST PARAMETER SUMMARY

Default type (r[read only], w[rite allowed], n[o_defaults]) : n
Are writes to other than the diagnostics cylinder allowed
                                                          : n
Are writes to the diagnostics cylinder allowed            : n
Number of errors allowed per device each subtest          : 0
Number of devices failed after which test aborts          : 12
Threshold for new flaws found during pattern test         : 0
Verbosity of test                                         : 3
If pattern testing, start after last completed pattern
                                                          : y
Enter OK, or :NN to return to question NN                 : OK

                      DRIVE CONFIGURATION DATA

       CCU  VME  Cntlr  Int Unit #   # Phys Log Prev
Drive   #    #   CSR  Level #  Cyl Hds Sec Sec Fmtd Name          Serial #
-----  ---  ---  -----  --- --- --- --- --- --- --- ------- ----------------
   1    7    0   0x200   2   0 1024  27  68  67  no DKD-208 00000
1000    7    0   0xe00   3   0 1024  27  68  67  no DKD-208 99999

Performing sysreset of ccus and memory...
Initializing I/O subsystem and Loading VIOP(s)...
```

# Hardware Initialization Sequence

After the last prompt is entered, and before test code execution, the following message is displayed during hardware initialization:

**Figure dev5130–5, Test Initialization Message**

```
Performing sysreset...
Initializing I/O subsystem and Loading VIOP(s)...
```

If writes to any of the disks are allowed, the following message is displayed immediately following the summary of inputs:

**Figure dev5130–6, Data Destructive Message**

```
**********************************************
* DATA DESTRUCTIVE OPERATION IS ABOUT TO START.*
* VERIFY WRITE-PROTECT IS SET ON ALL DRIVES    *
* EXCEPT THOSE YOU ARE TESTING/FORMATTING.     *
**********************************************

Then reconfirm that you want to continue [yn] ->
```

If **y** is entered then the actual testing commences with the loading of a driver into the VIOPs and then the execution of subtests begins. However, if **n** is entered, the test terminates and returns to the SPU prompt.

**Figure dev5130–7, Class 3 Test Initialization Message**

```
Subtest 300   0:00:00       Prepare for format
```

Before *dev5130* test code execution, the following events occur:

- A sysreset is performed
- Main memory is allocated for the test
- SPU windows to main memory are initialized
- SPU local test variables are initialized
- The VIOP is booted and loaded
- A driver on the VIOP is started
- VIOP local test variables are initialized

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

# Class Descriptions

The *dev5130* test contains four classes of subtests as shown in the following table:

## Table dev5130–6, *dev5130* Test Classes

| CLASS | DESCRIPTION |
|---|---|
| 1 | Controller functionality tests |
| 2 | Drive functionality tests |
| 3 | Drive system format tests |
| 4 | Interactive test |

The following sections describe the different classes and each of their subtests. Each section contains a table listing each subtest in that class and a description of the subtest. The Class 1 and Class 2 tests verify the controller functionality of the following:

- Reset through VMEbus backplane reset and through Command Status Register
- Used portion of short I/O space
- Invalid command detection and reporting
- Command set coverage includes the following:

| Tested commands | Untested commands |
|---|---|
| Diagnostics | Verify Sectors |
| Read Long | Reformat* |
| Write Long | Read Sectors Sequential* |
| Read Header | Write Sectors Sequential* |
| Read Raw Data | Verify Sectors Sequential* |
| Read CDC Flaw Map | Read Sequential, Disable Address Bump* |
| Report Configuration | Write Sequential, Disable Address Bump* |
| Write Sector Buffer | Verify Track |
| Read Sector Buffer | Verify Track Sequentially* |
| Report Sector ID | Extended Diagnostics* |
| Format with Sector ID | Dual Port Priority Select* |
| Initialize Long | Read and Scatter* |
| Report Configuration Long | Gather and Write* |
| Read Sectors | Read and Scatter 32* |
| Write Sectors | Gather and Write 32* |
| Verify Sectors | Read and Scatter 32 (list 2)* |
| Format Track | Read and Scatter 32 (list 2)* |
| Map Track | Buffer Allocation Check* |
| Handshake | Write After Cache* |
| Initialize | Clear Drive Fault |
| Restore | Map Sector |
| Seek | |
| Format Track with Data | |
| Read Noncached | |
| Track ID | |
| Fetch and Execute IOPB | |

* The commands marked with an asterisk were not available when this diagnostic was designed. They have been recently added. None of these commands are used by the *dev5130* test or by CONVEX UNIX. When a drive faults, the *Restore* command is used to clear the drive fault and rezero the unit instead of using the *Clear Drive Fault* command.

## Class 1 Subtests

Class 1 subtests consist of tests which exercise and verify most features of a controller. All of the tests are not data destructive since only the diagnostic cylinder is used for writing. Two options are given at the beginning of the test to specify whether writes to the diagnostic cylinder are allowed and whether writes to cylinders other than the diagnostics cylinder are allowed. These options control which subtests are allowed to run. The following table lists each Class 1 subtest and its description.

**Table dev5130–7, Class 1 Subtests**

| SUBTEST | DESCRIPTION | TIME (hr:min:sec) | DATA DESTRUCTIVE |
|---|---|---|---|
| 100 | Controller Reset | 0:00:05 | NO |
| 101 | Verify Diagnostics Commands | 0:00:02 | NO |
| 102 | Verify Product Identification | 0:00:02 | NO |
| 103 | Verify *Write Sector Buffer* and *Read Sector Buffer* Commands (max throttle) | 0:00:02 | NO |
| 104 | Verify Command Chaining | 0:00:03 | NO |
| 105 | Verify *Initialize* and *Initialize Long* Commands | 0:00:03 | NO |
| 106 | Verify *Format Track* Command and all combinations of interleaves 1 and 2 and skews 0 and 1 | 0:00:07[1] | YES[2] |
| 107 | Verify *Format Sector ID* and *Report Sector ID* Commands | 0:00:03[1] | YES[2] |
| 108 | Verify *Write Sectors* and *Read Noncached* Commands (all throttle settings) | 0:00:15[1] | YES[2] |
| 109 | Verify *Format Track with Data* Command | 0:00:02[1] | YES[2] |
| 110 | Verify *Write Long* and *Read Long* Commands | 0:00:04[1] | YES[2] |
| 111 | Verify *Read Sectors* Command | 0:00:05[1] | YES[2] |
| 112 | Verify *Map Track* and *Map Sector* Commands | 0:00:03[1] | YES[2] |
| 113 | Verify Controller Detects Bad IOPBs | 0:00:03 | NO |

[1] This time will double if two drives are connected to one controller.
[2] Set write protection for drive(s) to prevent loss of data.

### Subtest 100, Controller Reset

Subtest 100 performs a VMEbus chassis reset and verifies that the controller's Command Status Register (CSR) contains a 0x4000 pattern. Then the LED bit (0x8000) is set and read back to ensure it was set. The BDCLR bit in the CSR (0x2000) is set which resets the controller. Then the CSR is verified to indicate that it contains a 0x4000 pattern. The one bit that is on in this pattern is the board OK bit which is the controller's way of indicating it has passed its internal diagnostics.

### Subtest 101, Verify Diagnostics Commands

Subtest 101 performs each controller's diagnostics test using interrupts to respond to each controller's completion of a diagnostics command. The command's completion is verified by receiving an interrupt and checking that the board OK bit in the controller's CSR is set.

> **NOTE**
>
> The returned status and error codes for the remaining commands are checked in the Input/Output Parameter Block (IOPB) for correctness.

### Subtest 102, Verify Product Identification

Subtest 102 executes a controller *Handshake* command which returns the controller's product code, product variation, revision level, month, day, and year. This information is printed each time the subtest is run. If the information fails gross range-checking, the Input/Output Parameter Block (IOPB) indicates that an error has occurred, or if the release date of the firmware is older than the oldest acceptable firmware date, then the controller and all attached drives fail.

### Subtest 103, Verify *Write Sector Buffer* and *Read Sector Buffer*

Subtest 103 performs writes to and reads from one sector buffer in the controller's memory with an all '1's pattern and then an all '0's pattern. After the buffer is read, the data is compared. The throttle setting is set to 512 bytes per DMA burst which is the maximum amount of data that can be transferred in one controller *Write Sector Buffer* or controller *Read Sector Buffer* operation.

### Subtest 104, Verify Command Chaining

Subtest 104 verifies command chaining by performing chained controller *Write Buffer* and controller *Read Buffer* commands.

> **NOTE**
>
> The throttle setting is 64 bytes per DMA burst which is the size of the cache on the VIOP. This is the size of all DMA bursts in Subtests 104 though Subtest 400 except Subtest 108.

### Subtest 105, Verify *Initialize* and *Initialize Long*

Subtest 105 sets up a unit initialization block in VIOP memory and performs a controller *Initialize* command to load various device parameters into the controller. The subtest then uses the

controller *Report Configuration* command to verify the initialization. It then performs a controller *Initialize Long* command and verifies it using the controller *Report Configuration Long* command. All subsequent subtests use the *Initialize Long* command to initialize the controller. CONVEX UNIX also uses the *Initialize Long* command to perform controller initialization.

### Subtest 106, Verify *Format Track* and all Combinations of Interleaves

Subtest 106 executes the controller *Format Track* command to format all tracks on the diagnostic cylinder. The tracks are formatted with interleaves of 1 through 16 while using a skew of 0. Then skews of 0-15 are performed with interleave first set to 1 and then set to 2. Each of these 48 tests is verified by reading the track headers with the controller *Track ID Read* command and comparing the track headers to the expected headers.

### Subtest 107, Verify *Format Sector ID* and *Report Sector ID*

Subtest 107 verifies the controller *Format Sector Id* and controller *Report Sector Id* commands. First, track 0 of the diagnostic cylinder is formatted. Then a controller *Report Sector Id* command is performed on one track of the diagnostic cylinder and the headers are written out in reverse order using the controller *Format Sector Id* command. Then the format of the headers is verified by performing another controller *Report Sector Id* command and verifying the headers are reversed. Finally, the track is restored by another *Format Track* command.

### Subtest 108, Verify *Write Sectors* and *Read Noncached*

Subtest 108 verifies the controller *Write Sectors* and *Read Noncached* commands. First, a *Format Track* command is performed on one track on the diagnostic cylinder. Then the *Write Sectors* and *Read Noncached* commands are verified by performing writes to the minimum sector on the track. Then using the multiple sector technique, writes are performed to the minimum + 1 sector and minimum + 2 sector. Then reads are performed on the minimum sector using single sector I/O and then reads are performed on the next two sectors using multiple sector I/O. Next, a verify is performed to ensure that the data written was read back successfully. These operations (except for the format track) are repeated for each throttle setting (4, 8, 16, 32, 64, ... 512 bytes per transfer to and from main memory). If other controllers are available, the above steps are repeated on the other controllers.

### Subtest 109, Verify *Format Track with Data*

Subtest 109 verifies the *Format Track with Data* command. The *Format Track* command is used to format track 0 of the diagnostics cylinder by writing '0's to sector 0. Then the *Format Track with Data* command reformats the track with a `0x9abcdef0` repeating pattern. Then sector 0 is read and compared. The first four bytes of the sector are altered when formatted to reflect the cylinder, head and sector address of the sector. Then the first four bytes are verified.

### Subtest 110, Verify *Write Long* and *Read Long*

Subtest 110 verifies *Write Long* and *Read Long* commands by reading data and Error Correction Code (ECC) information from each drive and then writing the data and information back to the drive. Before beginning, the subtest performs a *Format Track* command if needed on track 0 of

the diagnostic cylinder. The minimum sector and the next sector are written to with test patterns 0x00000000 and 0xffffffff respectively and then the sectors are read using two *Read Long* commands. The headers and data fields are verified and the data and ECC information are swapped between the two sectors. The *Write Long* command is used to write the sectors back out and then the *Read Sectors* command is used to read the data back from the two sectors. The data should be in the opposite sectors corresponding to the sectors where the data was originally written.

### Subtest 111, Verify *Read Sectors*

Subtest 111 verifies the *Read Sectors* command. The *Write Sectors* command is used to write to the first two cylinders. Then the *Read Sectors* command is used to read the two cylinders. The subtest is repeated for all default patterns (refer to Subtest 301, Format and Pattern Test for patterns).

### Subtest 112, Verify *Map Track* and *Map Sector*

Subtest 112 verifies the *Map Track* and *Map Sector* commands. Two tracks on the diagnostic cylinder are formatted using the *Format Track* command and then sector 0 of the first track is mapped to sector 0 of the second track with the *Map Sector* command. Then the first sector is read and verified that the first word contains the address of the second sector in the first word (written by the *Format Track* command).

Then the two tracks are reformatted using the *Format Track with Data* command and the first track is mapped to the second track using the *Map Track* command. Then the sectors on the first track are read. The first word of each sector should contain the address from the second track (written by the *Format Track with Data* command).

Next, the first track is reformatted.

### Subtest 113, Verify Controller Detects Bad IOPBs

Subtest 113 verifies the controller is capable of detecting bad Input/Output Parameter Blocks (IOPBs). A head address outside the controller's limit is sent to the controller for checking. The controller should reject the address. Then a cylinder address outside the drive's range is sent to the controller. The controller should also reject the cylinder address

## Class 2 Subtests

Class 2 subtests consist of tests which exercise drives to verify that they are mechanically and electrically functional.

The Class 2 subtests verify the drive functionality of the following:

- Write and read every head
- Single and multi-track seeks

- Fault condition detection

The following table lists each Class 2 subtest and its description.

**Table dev5130–8, Class 2 Subtests**

| SUBTEST | DESCRIPTION | TIME[3] (hr:min:sec) | DATA DESTRUCTIVE |
|---|---|---|---|
| 200 | Verify Head Switching | 0:00:42[1] | YES[2] |
| 201 | Verify Sequential-Track Seeks | 0:00:30[1] | YES[2] |
| 202 | Perform Min-to-Max Track Seeks | 0:00:20[1] | NO |
| 203 | Perform Accordion Seeks | 0:01:00[1] | NO |
| 204 | Perform Random Seeks | 0:01:00[1] | NO |
| 205 | Verify Detect of Forced Faults (cylinder and head) | 0:00:30[1] | NO |

[1] This time will double if two drives are connected to one controller.
[2] Set write protection for drive(s) to prevent loss of data.
[3] Times shown are for one Hitachi DK514-38 drive.

### Subtest 200, Verify Head Switching

Subtest 200 verifies head switching and also verifies that each head can write and read. Special formatting takes place for Class 2 subtests, if necessary, before the main portion of this subtest executes. The controller *Write Sectors* command is used to write to each sector of each track on the diagnostic cylinder. A controller *Read Noncached* command is used to read the data and then the write and read buffers for each sector are compared. First a `0x0000` pattern is used and followed by a `0xffff` pattern. The *Write Sectors* command and the *Read Noncached* command, and the compare are repeated ten times.

### Subtest 201, Verify Sequential-Track Seeks

Subtest 201 verifies sequential-track seeking by first formatting head 0 on all cylinders (if writing is allowed). Then a controller *Report Sector ID* command is performed on head 0 of each cylinder starting with cylinder 0 to verify positioning.

### Subtest 202, Perform Min-to-Max Track Seeks

Subtest 202 performs min-to-max track alternate seek. Special formatting takes place for Class 2 subtests, if necessary, before the main portion of this subtest executes. A controller *Track ID Read* command performs an implied seek to the minimum track. After the test verifies the cylinder number, a controller *Track ID Read* command performs an implied seek to the maximum track. When this cylinder is verified, the seek repeats to the minimum and maximum track 100 more times with verification at each track. Then the seek repeats, using the controller *Seek* command with no verification until the seek is complete. To complete the test, a controller *Track ID Read* command is used to read the last track and positioning is verified.

### Subtest 203, Perform Accordion Seeks

Subtest 203 performs an accordion seek on a drive. Special formatting takes places for Class 2 subtests, if necessary, before the main portion of this subtest executes. A controller *Track ID Read* command performs an implied seek to the minimum track, maximum system track, min+1, max-1, min+2, max-2, etc., until a one-track seek is performed. The test verifies the seek by checking each of the headers just read. The accordion seek then repeats using the controller *Seek* command with no verification. After the last seek is performed, a controller *Read Header* command is performed to verify the final positioning. The test chains commands to two drives on a controller if both drives are connected to one controller.

### Subtest 204, Perform Random Seeks

Subtest 204 performs 1000 random seeks within the specified minimum and maximum cylinders on a drive. Special formatting takes places for Class 2 subtests, if necessary, before the main portion of this subtest executes. Random numbers are created by initializing C's random number generator (*rand*) with a seed value of 113 and then calling *rand* for successive numbers. The controller *Track ID Read* command is used for implied seeks which verify positioning. The test then performs the random seek using the controller *Seek* command with no verification until the seeks are complete. Then a controller *Read Header* command is used to verify the last cylinder. The test chains commands to two drives on a controller if both drives are connected to one controller.

### Subtest 205, Verify Detect of Forced Faults

Subtest 205 verifies the ability of the SMD drives to detect fault conditions. A controller *Initialize* command is used to set the head and cylinder limits to two greater than the maximum values specified in the file */mnt/bin/lib/DB_diskfmt*. Then a controller *Read Sectors* command is used to read a track from a cylinder that is two greater than the maximum. This generates a seek error. An error may not occur if the number of cylinders specified in */mnt/bin/lib/DB_diskfmt* is incorrect or if the drive has exactly 1023 or 1024 cylinders. The subtest expects certain errors or no errors since all known drives are characterized. For future drives, an `Expected error did not occur` error should be displayed if a drive fault or seek error does not occur.

Next, the test sets the head to two greater than the maximum value and then a controller *Write Sectors* command is used to write to this head on the diagnostic cylinder. This generates a write fault. After the write fault, the error does not clear until a drive reset is performed. The test performs a read of head 0 of the diagnostic cylinder and verifies that the write fault still exists. Then the test resets the drive and reads head 0 again; the write fault should no longer exist.

## Class 3 Subtests

Class 3 subtests consist of tests which format new drives or reformat previously formatted drives for system use. The following table lists each Class 3 subtest and its description.

## Table dev5130–9, Class 3 Subtests

| SUBTEST | DESCRIPTION | TIME (hr:min:sec) | DATA DESTRUCTIVE |
|---|---|---|---|
| 300 | Format Setup | N/A | YES[3] |
| 301 | Format and Pattern Test | 2:30:00[1] | YES[3] |
| 302 | Fix Flaws | 0:01:00 | YES[3] |
| 303 | Initialize Diagnostic Cylinder | 0:04:00[2] | YES[3] |
| 304 | Verify System Format | 0:08:00 | NO |
| 305 | Verify Diagnostic Cylinder | 0:00:30[2] | NO |
| 306 | Verify Pattern Test Error Threshold | 0:00:02 | NO |

[1] Time varies depending on the types of drives and number of drives per controller. For one drive, the times are the following:

NEC 2352 (1/2 GByte) -> 1.25 hours
NEC 2363 (1 GByte) -> 2.30 hours
Hitachi 514-38 (RDS drive) -> 1.00 hours

Two drives on one controller will double these times.
If drive types are mixed, use the longer time.

[2] Multiply this time by the number of drives being formatted.
[3] Set write protection for drive(s) to prevent loss of data.

### Subtest 300, Format Setup

Subtest 300 provides an interactive means of changing the way disks are formatted and also allows the user to input/edit flaw location information.

First, the subtest attempts to read the Manufacturer's Defect (MD) list and Grown Defect (GD) list on the innermost cylinder. If a MD list is not available, then the subtest sees if the file */mnt/usr/backup_maps/bkup.s* exists where *s* is the last five characters of the drive's serial number. If so, this copy is used.

First, the following initialization message for Subtest 300 is displayed:

## Figure dev5130–8, Subtest 300 Initialization Message

```
Subtest 300   0:00:00       Prepare for format
```

Next, the following interactive prompt is displayed:

## Figure dev5130–9, Subtest 300 Interactive Prompt

```
Type 'h' for help with commands
format setup (D1;Defect)->
```

If **h** is entered, a help facility is displayed which lists the available commands. The help display is shown in the following figure:

## Figure dev5130–10, Subtest 300 Help Screen

```
Commands:
q[uit]                      - terminate the test and return to UNIX prompt
c[hange_mode]               - toggles logical sector/defect map entry mode
co[ntinue]                  - exit setup and continue with format
cyl hd sec                  - logical sector to be slipped
cyl hd bcai len             - defect map entry to be slipped
de[lete] cyl hd sec         - delete logical sector entry
de[lete] cyl hd bcai len    - delete defect map entry
d[rive] [n]                 - change/display current drive
f[ile] filename             - get inputs from specified file
fo[rmat_options]            - allows you to modify the format
h[elp]                      - display this information
l[ist] [{m,g,n,needs}]      - list all, manu[m], grown[g], new[n], needs
m[ap_track] cyl hd          - maps a track to an alternate
no_f[law_map]               - to inform formatter that no flaw map exists
no_t[rack_flaw] cyl hd      - used when no flaw data exists for a track
!unix-command               - execute unix command
'comment                    - comment; echoed to screen
```

### Subtest 300, *change_mode* Command

**c[hange_mode]**

When manually entering defects, first select the entry mode. The default mode is Byte-Count-After-Index (BCAI) mode. To enter defects as logical sectors, modes must be switched with the *change_mode* command. Then enter the position of each flaw. In sector mode, this means typing the cylinder, head, and sector where the flaw exists. In BCAI mode, enter the cylinder, head, byte-count-after-index, and bit-length.

The input mode (logical sector or defect map mode) is revealed in the slip prompt as shown in the following figure:

## Figure dev5130–11, Changing Input Modes

```
format setup (D1;Sector)->c
format setup (D1;Defect)->
```

### Subtest 300, *continue* Command

**co[ntinue]**

The *continue* command is used to exit setup and continue with the format. When all inputs have been entered, type **continue**. Before Subtest 300, Format Setup exits, it will read the original Manufacturer's Defect (MD) maps from all drives for which no MD lists already existed and for which no backups existed.

Any drives which had no MD list on the innermost cylinder or backup file but did have original MD maps now have backup files containing the data from the original MD maps. The name of the files are */mnt/usr/backup_maps/bkup.s* where *s* is the last 6 digits of each drive's serial numbers.

For those drives that did not have a valid MD list or GD list, the defect cylinder (the cylinder where MD and GD lists are recorded) is formatted, pattern-tested and flaws remapped. Then the MD and GD lists for these drives are written to the drives. Subtest 300 then exits.

Refer to the *list*, *no_flaw_map*, and *no_track_flaw* commands for more information.

### Subtest 300, *cyl hd bcai len* and *cyl hd sec* Commands

*cyl hd bcai len*

or:

*cyl hd sec*

The *cyl hd bcai len* command allows a defect map entry to be slipped. The *cyl hd sec* command allows a logical sector to be slipped. When a defect is entered, a list of logical sectors that correspond to the defect is generated as follows:

## Figure dev5130–12, List of Logical Sectors For a Defect

```
format setup (D1;Defect)-> 293 1 572 9510
        Associated logical sectors: 59 0 1
```

The figure illustrates that three sectors are affected by the media flaw which starts at byte 572 from index and is 9510 bits long.

If two defects exist that affect the same sector, the logical sector will only be entered once. Subsequent entries of the same sector will result in the message:

```
This entry already exists (ignored)
```

For instance, given the following two defects:

| cylinder | head | bcai | bit-length |
|---|---|---|---|
| 10 | 0 | 70 | 1 |
| 10 | 0 | 80 | 1 |

Both defects fall in sector 0 so when the second defect is entered, it will not create a new entry in the input list.

### Subtest 300, *delete* Commands

**de[lete]** *cyl hd sec*

or:

**de[lete]** *cyl hd bcai len*

The *delete* command is used to delete a logical sector entry or a defect map entry. An example of entering a bad location and deleting it is shown along with the *list* command in the following figure:

## Figure dev5130–13, Entering and Deleting an Incorrect Location

```
format setup (D1;Sector)-> 17 7 23
format setup (D1;Sector)-> 17 7 46
format setup (D1;Sector)-> del 17 7 46
        Input 17 7 46 has been removed
format setup (D1;Sector)-> list

Summary of flaws on drive 1 serial number 003013
 CYL HD SEC    BCAI    LEN TYPE  ERR SRC | CYL HD SEC    BCAI    LEN TYPE  ERR  SRC
  17  7  23   35032   4825       NONE USR |
```

As shown in the previous figure, use the *list* command to display inputs and defect list entries and with the *delete* command delete any incorrect entries except MD list entries (refer to the *list* command in Subtest 400, Interactive Test for more information about listing flaws). MD list entries are marked with MAP in the SRC column of a list.

### Subtest 300, *drive* Command

**d[rive]** [*n*]

The *drive* command is used to change or display the current drive. If a drive is not specified, a list of all selected drives is displayed and a prompt is displayed to enter a drive number. An example of switching drives follows:

**Figure dev5130–14, Switching Drives Using the *drive* Command**

```
format setup (D1;Sector)-> d 2
        Drive 2 is now selected
```

### Subtest 300, *file* Command

**f[ile]** *filename*

To enter the defect data from one or more files, select each file by typing:

**file** *filename*

where *filename* is the actual name of the file. The first line of the file defines the drive type and looks like this:

n *drivename*

where *drivename* is the drivename which can be found in */mnt/bin/lib/DB_diskfmt* on the Service Processor (for example, DKD-208 for the 1 GByte NEC 2363 drive). Following the drive name line, a file has remaining lines contain either BCAI inputs or logical sector inputs. A BCAI input is preceded with the letter **d** while a logical sector input is preceded with the letter **s**. The spare sector can be specified by using a logical sector number one greater than the last used sector number.

To specify a track that is to be mapped to an alternate track, use the letter **m** followed by the cylinder and head of the bad track. One or more spaces separate each item in the file and a new line can be started between any two items. To include comments in the file type a # at any place on a line. Everything from the # to the end of the line is ignored. Blank lines are also allowed. A file containing defect data is shown in the following figure:

## Figure dev5130–15, Creating a Defect Data File

```
#------------------------------ start of file -----------------------------
# serial number: 003013
n  DKD-206
#   cyl  hd   bcai  len       cyl  hd   bcai  len  # NEC 2352 drive
d   157  15  24395   34    d  297  14  14102    4  # BCAI Inputs
d   466   0    206    1

#   cyl  hd  sec             cyl  hd  sec
s   457  12   42          s  821   5   11         # Logical sector inputs

#   cyl  hd
m   622  14                                      # map track input
#------------------------------ end of file -----------------------------
```

Defect input mode only applies to manual inputs. File inputs are controlled by the character that precedes the defect location.

### Subtest 300, *format_options* Command

**fo[rmat_options]**

The *format_options* command is used to modify the format of the drive. To manipulate usage of the MD list or GD list, type **format_options**. Several prompts must be answered in response to this command. The prompts are displayed in the following figure:

## Figure dev5130–16, Prompt Display for the *format_options* Command

```
For Drive 1 on controller (viop/vb/csr) 7/0/e00:
     Wipe out defect lists if they exist [yn] (n) -> RETURN
     Use manufacturer's defects during slip of sectors [yn] (y) -> RETURN
     Keep old grown defects  [yn] (y) -> RETURN
               - if 'n', next question will be asked
     About to delete grown defects. Continue  [yn] (y) -> RETURN
```

The defaults are in effect unless changed.

> **NOTE**
>
> If a drive is being reformatted, the use of the Manufacturer's Defects will be the same as the last time it was formatted unless this command is used to change them.

### Subtest 300, *help* Command

**h[elp]**

This command lists the commands and their arguments.

### Subtest 300, *list* Command

**l[ist]** [{**m,g,n,needs**}]

The *list* command is used to list all manufacturer defects, grown defects, new defects, or defect needs. If the currently-selected disk has been previously formatted, then the MD list and GD list will already be read when the first prompt appears. This data can be listed using the *list* command. If the drive is unformatted, the original manufacturer's defect map is read when exiting setup. If defect needs were listed after the *continue* command, type **list needs** to redisplay the current needs.

## Figure dev5130–17, Using the *list* Command

```
format setup (D2;Sector)-> list

Summary of all flaws on drive 2 serial number 003418
 CYL HD SEC   BCAI   LEN TYPE  ERR SRC | CYL HD SEC   BCAI   LEN TYPE  ERR SRC
 100  5  52  10536     1       NONE USR | 257  9  35  26401  1392       NONE USR
 257  9  58  26576   600       NONE USR |

format setup (D2;Sector)->
```

### Subtest 300, *map_track* Command

**m[ap_track]** *cyl hd*

To map an entire bad track, use the *map_track* command. The test selects one of the alternate tracks for use.

### Subtest 300, *no_flaw_map* and *no_track_flaw* Commands

**no_f[law_map]**

or:

**no_t[rack_flaw]** *cyl hd*

The *no_flaw_map* command is used to inform the formatter that no flaw map exists. The *no_track_flaw* command is used when no flaw data exists for a track. After using the *continue* command, if no defects were readable from anywhere or some of the original MD map was unreadable, then the user is prompted to enter the needed defect data. These inputs can come from a file of defects or from manual inputs. If inputs do not exist to satisfy the listed needs, then use one of the two commands to satisfy the test. To cancel a need for inputs due to an unreadable map type **no_flaw_map**. To cancel a need for a specific track type **no_track_flaw**

*cyl hd* where *cyl hd* is the track for which a need was listed.

### Subtest 300, *quit* Command

**q[uit]**

The *quit* command is used to terminate the test and return to the UNIX prompt. The prompt in the following figure is displayed when the **quit** command is issued:

**Figure dev5130–18, Executing the *quit* Command Within Subtest 300**

```
format setup (D2;Sector)->quit
        Are you sure you want to terminate the test [yn] (y) ->
```

### Subtest 300, Execute UNIX Command

! *UNIX Command*

The *!* (UNIX command) is used to execute a UNIX command while within Subtest 300. An example of its use is displayed in the following figure:

**Figure dev5130–19, Executing a UNIX Command Within Subtest 300**

```
format setup (D2;Sector)-> !cat def_003418     <-to illustrate unix cmd
# Serial Number: 003418
n DKD-206
d  100  5 10536     1
d  257  9 26401  2000
```

### Subtest 300, Enter Comments

*'comment*

This command is used to enter a comment. To enter a comment, type the ' command and a comment. The comment is echoed so it is redirected output. This is useful to document redirected output.

### Subtest 301, Format and Pattern Test

Subtest 301 formats a drive and pattern tests all sectors except the ones in the defect list cylinder. This includes sectors that will later be used as spares. The patterns used for pattern testing are

listed in the following table:

**Table dev5130–10, Patterns for Pattern Testing**

| SET | PATTERN | |
|---|---|---|
| 1 | MFM Data Encoding | |
| | 0x00000000 | 0x6DB66DB6 |
| | 0xFFFFFFFF | |
| | 0x55555555 | |
| | 0xAAAAAAAA | |
| 2 | 2-7 RLL Data Encoding | |
| | 0x00000000 | 0x9ABCDEF0 |
| | 0xCCCCCCCC | 0xAF5BAF5B |
| | 0x88888888 | 0x22222222 |
| | 0xBCDEF09A | |
| 3 | 1-7 RLL Data Encoding | |
| | 0x00000000 | 0x9ABCDEF0 |
| | 0xCCCCCCCC | 0xAF5BAF5B |
| | 0x88888888 | 0x22222222 |
| | 0xBCDEF09A | |
| 4 | Combination of Unlike Drives | |
| | 0x6db66db6 | 0xffffffff |
| | 0x9abcdef0 | 0xcccccccc |
| | 0x00000000 | 0x33333333 |
| 5 | User Specified Patterns | |
| | Patterns in *dev5130.patt* file | |

If all drives that are to be formatted use the same encoding scheme, then the corresponding set of patterns will be used. However, if drives with different encoding schemes are formatted together, then pattern set 4 is used.

These default patterns can be replaced by up to 15 other patterns if the file *dev5130.patt* is created in the current directory or if it is created in */mnt/bin/lib*. The current directory is searched first. Each pattern is preceded with a 'p' and at least one space, tab or newline character. The pattern can be from 1 to 1024 hex characters per pattern (upper or lowercase letters are allowed). More than one line can be used to define the patterns. Blank lines are also allowed.

The following is an example of two patterns in a *dev5130.patt* file:

```
p A    p 23456789a
```

The first pattern is expanded so that every byte written contains a 0xAA pattern. The second pattern has 9 hex digits which are repeated when written. The following is the resulting data:

```
23456789 a2345678 9a234567 89a23456  789a2345 6789a234 56789a23 456789a2
                  ---------- etc ----------
```

Any sector-related errors (header errors; hard Error Correction Code (ECC) error; or correctable ECC error) are retried immediately and, if an error occurs again in ten retries (does not have to be the same error), the sector location is saved for later slipping. Errors that do not repeat are discarded. A maximum of 2048 errors can be saved per drive. This includes the flaws from Subtest 300, Format Setup and any found during pattern test. If a verbose value of 16 is selected during test parameter inputs, each error is printed at the time it is detected. A verbose value of 2 results in only grown defects being printed. The last operation in Subtest 301, Format and Pattern Test writes the MD and GD lists.

### Subtest 302, Fix Flaws

If the innermost cylinder does not yet contain the new Manufacturer's Defect (MD) list, then the drive fails.

If Subtest 301 Format and Pattern Test was not run, then the drive is formatted with one spare sector per track. Then all sectors which have been marked bad will be slipped. If more than one sector is bad on a track or a header-related error occurred on the track, the entire track is remapped to an alternate track. The number of cylinders worth of spare tracks for a particular drive is defined in the *DB_diskfmt* file.

If verbose value 1 is selected during the test parameter menu (this is the default along with printing each grown defect as it is found), a summary of bad sectors and tracks for each drive will be printed before slipping occurs.

The last operation writes the MD and GD lists.

### Subtest 303, Initialize Diagnostic Cylinder

Subtest 303 writes special patterns to the diagnostic cylinder which is the cylinder just before the defect list cylinder. Sector zero of each track contains a table of contents describing what is written to each of the remaining sectors. If no sectors have been relocated on a track, sector 1 is marked bad. The next twelve sectors contain Error Correction Code (ECC) errors of length 1 to 12 respectively. The remaining sectors have the patterns listed in Subtest 301, Format and Pattern Test written to them. Sector 13 (or 14 if sector 1 is marked bad) will contain the first pattern, sector 14 (or 15) will contain the second pattern, and so on with the patterns repeating throughout the remaining sectors of each track.

### Subtest 304, Verify System Format

Subtest 304 reads all five copies of the original Manufacturer's Defect (MD) list, Grown Defect (GD) list, and track map log and verifies that at least three of each are good. If not, the test fails.

Then the subtest reads every logical sector in the usable portion of the disk. Any errors are reported since all bad sectors should be relocated and be handled by the controller.

Any errors are reported and do count against the device errors per subtest limit. If the number of errors exceeds the limit, the drive is failed. The usable portion of each drive is all cylinders preceding the alternate track area and any used alternate tracks.

### Subtest 305, Verify Diagnostic Cylinder

Subtest 305 reads each sector on the diagnostic cylinder and expects soft or hard Error Correction Code (ECC) errors on sectors 1 through 12 of each track (or sectors 2 through 13 if sector 1 was marked bad). The remaining sectors are read with data compare and must be error-free.

### Subtest 306, Verify Pattern Test Error Threshold

A threshold can be set during user prompts which will determine how many new errors can be detected during Subtest 301, Format and Pattern Test pattern testing before this subtest fails. The intention is that all errors detected by pattern test should already be in the MD map. If the errors are actually new, this may indicate an unacceptable degradation of the disk since it was tested at the manufacturer or inaccuracy in manufacturer's media tester.

## Class 4 Subtest

The Class 4 subtest consists of one test, the interactive test, which allows the user to read and display sectors, verify parts of disks, format single tracks, slip sectors, etc.

**Table dev5130–11, Class 4 Subtest**

| SUBTEST | DESCRIPTION | TIME (hr:min:sec) | DATA DESTRUCTIVE |
|---|---|---|---|
| 400 | Interactive Test | N/A[1] | YES[2] |

[1] The time for the interactive test depends on the operations performed.
[2] Set write protection for drive(s) to prevent loss of data.

### Subtest 400, Interactive Test

Subtest 400 provides a special interactive test interface to allow the user to perform a variety of disk maintenance tasks such as pattern-testing portions of the disk, slipping bad sectors, formatting single tracks, and several other useful operations. Help information is available when needed.

> **CAUTION**
>
> Five of the commands are potentially data destructive: *pattern_test, slip_sectors, format, data_verify,* and *copy_from_spu.*

The first four commands do an automatic save of the data the command is going to overwrite before the operation begins with the following exceptions:

- An attempt to pattern test more than one track is performed and **y** is entered when asked to confirm multiple track pattern test (because only one track can be saved).
- An attempt to preserve a track's data before a destructive operation fails and a response is entered to force the operation anyway.
- The pattern test, slip sectors, or format operation has completed and then data errors occur during the restore of the track's data.
- The keys CTRL and **c** or CTRL and **b** are pressed in the middle of the operation.
- Automatic saving is disabled by using the command *toggle_save*.
- A power outage or some other unforeseen event causes the test to abort in the middle of the operation.

The fifth command, *copy_from_spu* restores data to the drive which has been saved using *copy_to_spu*. Since it writes to the disk, it can be destructive. However, the usual way this could corrupt data is if the temporary file created by *copy_to_spu* is edited before using *copy_from_spu* to restore the data.

### Subtest 400, Interactive Test Invocation

To invoke the *dev5130* interactive test, use the procedure shown in the following figure. All responses in **boldface** are entered by the user. The [+> *filename*] option appends all test results to file *filename*. The prompts and responses in the following figure would appear sequentially on the screen, one line at a time. All the prompts and responses are shown in one figure for convenience.

To invoke the test, use the procedure shown in the following figure:

## Figure dev5130–20, Subtest 400 Interactive Test Invocation Sequence

```
(spu)>sysreset; mminit -s; dshell RETURN

CONVEX DIAGNOSTIC SHELL

: test dev5130 -s 400 [+> filename] RETURN
```

### Subtest 400, Interactive Test Menu

Once the test is invoked, prompts are displayed to specify various options. The following figure shows typical 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.

## Figure dev5130–21, Subtest 400 Interactive 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
Default type (r[read only], w[rite allowed], n[o_defaults])
[w,r,n]                                              (r) ->
                  PERIPHERAL CONFIGURATION DATA
       CCU      Chassis   Type      CSR     Int  Unit  Type
      -------  -------  --------  -----  ---  ----  -------
 1)   viop 3      0      DKC-204   0xc00   1     0    DKD-206
 2)   viop 3      0      DKC-204   0xa00   4     0    DKD-208
 3)   viop 6      1      DKC-203   0xc00   3     0    DKD-214

Enter device 99 to begin user-defined configurations or -1 to end selection
 2:  Device selection [-1,1-3,99]                        (-1) ->1
 3:  Previously formatted on CONVEX system with Interphase 4200/4201 controller
     [y,n]                                               (n) ->y
 4:  Serial number (5 to 31 characters) []
                          (Default: read from drive) ->RETURN
 5:  Device selection [-1,1-3,99]                        (-1) ->2
 6:  Previously formatted on CONVEX system with Interphase 4200/4201 controller
     [y,n]                                               (n) ->y
 7:  Serial number (5 to 31 characters) []
                          (Default: read from drive) ->RETURN
 8:  Device selection [-1,1-3,99]                        (-1) ->3
 9:  Previously formatted on CONVEX system with Interphase 4200/4201 controller
     [y,n]                                               (n) ->y
10:  Serial number (5 to 31 characters) []
                          (Default: read from drive) ->RETURN
11:  Device selection [-1,1-3,99]                        (-1) ->RETURN
12:  Enter OK, or :NN to return to question NN [OK]
                                                        (OK) ->RETURN

*********************************************
* DATA DESTRUCTIVE OPERATION IS ABOUT TO START.*
* VERIFY WRITE-PROTECT IS SET ON ALL DRIVES    *
* EXCEPT THOSE YOU ARE TESTING/FORMATTING.     *
*********************************************

Then reconfirm that you want to continue [yn] ->y
```

As shown in the previous figure, enter **y** to the first prompt. Then select a drive to test at the `Device selection` prompt and always enter **y** to the `Previously formatted` prompt and always enter RETURN to the `Serial number` prompt. This drive selection process may be repeated for additional drives.

When all drives to be tested have been selected, press RETURN twice to select the defaults to the `Device selection` and `Enter OK` prompts. Then enter **y** to the `Then confirm` prompt to allow the test to continue. After a minute, the interactive test will start up.

### Subtest 400, Interactive Test Mode

Once test initialization is complete (typically takes less than one minute), the following is displayed:

**Figure dev5130–22, Subtest 400 Interactive Test Mode**

```
                INTERACTIVE TEST MODE
                FOR SMD AND ESDI DRIVES

Type 'h' for help
(D1;Save)->
```

**NOTE**

D1 means drive 1 is selected and Save means that save of track data is automatic during the destructive commands *format*, *pattern_test*, *data_verify* and *slip_sectors*.

Type **h** at any time to display the list of available commands.

To exit the interactive test, type **q** and press RETURN. If a track is saved, a prompt asks if it is okay to lose this data. If **y** is entered or if a track is not saved, *dev5130* ends normally and exits back to the *dshell* prompt. If **n** is entered, test execution returns to the interactive prompt with nothing changed.

**NOTE**

When the interactive test is exited, the drives under test will ALWAYS show PASSED in the summary. This is because all failures are ignored during the interactive test.

### Subtest 400, Interactive Test Commands

The following table lists all the available interactive test commands. Also a brief description of each command's purpose is listed. For the exact syntax of each command, refer to the command's description in the following section.

**Table dev5130–12, Subtest 400 Interactive Test Commands**

| COMMAND | DESCRIPTION |
|---|---|
| **a[lternate_seek]** | Seek between two selected cylinders |
| **copy_f[rom_spu]** | Restore sectors from service processor disk to SMD |
| **copy_t[o_spu]** | Write sectors from SMD to service processor disk |
| **da[ta_verify]** | Write once, repetitively read and compare |
| **dr[ive_select]** | Select drive |
| **du[mp_lists]** | Dump defect lists to service processor disk |
| **fo[rmat]** | Reformat one track |
| **hd[e_display]** | Display header, data, and ECC |
| **h[elp]** | Display list of commands and arguments |
| **l[ist]** | Display list of flaws |
| **pa[ttern_test]** | Pattern test range of sectors |
| **pr[otection]** | Enable or disable write protection |
| **q[uit]** | Exit Interactive Test |
| **res[tore_track_data]** | Restore previously saved track data |
| **sa[ve_track_data]** | Save one track of data |
| **sec[tor_display]** | Display data from range of sectors |
| **ser[ial_number]** | Display or change disk serial number |
| **sl[ip_sectors]** | Slip one or more sectors |
| **st[atus]** | Display information about device |
| **to[ggle_save]** | Enable and disable automatic save |
| **tr[ack_headers_display]** | Display track headers |
| **v[erify_format]** | Verify a range of sectors |
| **!***UNIX-command* | Execute UNIX command |
| **'***comment* | Comment; ignored |

Each command can be executed by entering the portion of the command that precedes the brackets or by entering all of the command. For example, the *quit* command can be executed by entering **q** or **quit**. The following sections describe the different interactive test commands.

### Subtest 400, Seek Between Two Selected Cylinders

**a[lternate_seek]** [*nnn* **times**] *min-cyl* **to** *max-cyl*

This command provides the capability of performing alternate seeks between two cylinders so adjustments can be made to drives. Seeks are between the two specified cylinders to head. Data is not transferred during the seeks.

Arguments:

- [*nnn* **times**] — Repetition count. Possible values include:
  - **1** — Lower bound value, default value
  - **2,147,483,647** — Upper bound value
- *min-cyl* **to** *max-cyl* — The seeks occur between these two cylinders.

### Subtest 400, Restore Sectors From Service Processor Disk to SMD

**copy_f[rom_spu]** [*filename*]

This command restores sectors previously stored to the service processor disk by the *copy_to_spu* command. If a filename is not specified, the default file created by *copy_to_spu* (filename */mnt/usr/backup_maps/save.s* where *s* is the serial number) is used. This is the file that *copy_from_spu* attempts to read if a file is not specified.

Since the service processor file is in an editable (ASCII) format, the data or the cylinder, head, and sector where it is to be restored can be edited before performing the *copy_from_spu* operation.

### Subtest 400, Write Sectors From SMD to Service Processor Disk

**copy_t[o_spu]** [*filename*] *cyl hd sec* [**to** *cyl hd sec*]

This command allows the saving of one or more sectors to the service processor disk in an editable (ASCII) format.

The *copy_from_spu* command restores these sectors. If a filename is not specified, the default file (filename */mnt/usr/backup_maps/save.s* where *s* is the drive's serial number) is used. The save file can be edited to change the data in the sectors or to change where the sectors will be restored. If a file is chosen that already exists, this command overwrites the file.

Arguments:

- *cyl hd sec* — Starting sector
- [**to** *cyl hd sec*] — Ending sector, default is starting sector

### Subtest 400, Write Once then Repetitively Read and Verify

**da[ta_verify]** [*nnn* **times**] *cyl hd sec* [**to** *cyl hd sec*] [**with** *pattern*]

This command writes to all the sectors specified with the first default pattern or the pattern specified on the command line. It then rereads the data *nnn* times (or one time by default) without data compare. Only errors detected by the drive or controller are reported. In the list of arguments, [**to** *cyl hd sec*] may be substituted with [**to end**].

Arguments:

- [*nnn* **times**]— Repetition count. Possible values include:
  - **1** — Lower bound value, default value
  - **2,147,483,647** — Upper bound value
- *cyl hd sec* — Starting sector
- [**to** *cyl hd sec*] — Ending sector, default is starting sector
- [**with** *pattern*] — The *pattern* is a 1 to 8 hexadecimal digit pattern

### Subtest 400, Select Drive

**dr**[**ive_select**] [*drive-number*]

This command can change which drive is the active drive. Most of the other interactive test commands operate on the active drive only. The argument *drive-number* is the entry number from the *Drive Configuration Data* which is displayed during the test parameter entry. To redisplay the entry numbers enter **d** and press (RETURN). The *Drive Configuration Data* is redisplayed along with which drive is currently active. Then a new drive selection prompt is displayed, and an entry number can be entered to change the active drive or pressing (RETURN) leaves it unchanged.

Argument:

- [*drive-number*] — The entry number from the *Drive Configuration Data* which is displayed during the test parameter entry, the active drive

The following is an example of displaying the drive configuration data and selecting a drive:

**Figure dev5130–23, Displaying Drive Data and Selecting a Drive**

```
(D1;save)-> d

                        DRIVE CONFIGURATION DATA
      CCU VME Cntlr Int Unit #   # Phys Log Prev
Drive  #   #   CSR Level #  Cyl Hds Sec Sec Fmtd Name          Serial #
----- --- --- ----- --- --- --- --- --- --- --- ------- ----------------------
   1   3   0  0xc00  1   0  760  19  60  59 yes DKD-206 03991
   2   3   0  0xa00  4   0  760  19  60  59 yes DKD-206 03268
   3   6   1  0xc00  3   0  842  20  46  45 yes DKD-208 04123

        Drive 1 is currently selected
        Enter new selection or 0 to leave it as is (0,1-3) [0] -> 2
        Drive 2 is now selected
(D2;save)-> d 3
        Drive 3 is now selected
(D3;save)->
```

The configuration data is not displayed if the drive number is entered with the command. Also, the previous figure shows that the drive number is redisplayed with each prompt as D*x* where *x* is the current drive number.

### Subtest 400, Dump Defect Lists to Service Processor Disk

**du[mp_lists] [b][m|g|a]**

This command is used to create a backup of the defect lists for the following reasons:

- Before moving a drive to a controller other than the Interphase 4200 SMD controller
- Whenever the disk is going to be completely overwritten for security reasons

The file can be dumped in ASCII or binary. An ASCII file is essential if the purpose is to move the drive to a Multibus system since the Multibus drive formatter can only read ASCII files to date. The binary dump capability is primarily for future use to allow a dump of the defect lists so the lists can be restored with no changes. When restoring an ASCII file, all entries become grown entries which means the information is lost as to whether the flaw originated with the manufacturer or later appeared as a grown defect. Also, a binary dump potentially occupies less room on the service processor disk.

Arguments:

- [b] — Dump file in binary (if not specified, dump file in ASCII)
- [m] — Dump only Manufacturer's Defects
- [g] — Dump only Grown Defects
- [a] — Dump all defects, default value

### Subtest 400, Reformat One Track

**fo[rmat]** *cyl hd*

This command is used to unslip sectors, unmap a track, or recreate a bad track's headers. If automatic save is enabled (the prompt indicates *Save*), then before the track is reformatted, an attempt is made to read every logical sector (including relocated sectors). If a read fails, the options of retrying the sector, terminating the format before any destructive operation occurs, or forcing the format even though one or more sectors may not be recovered are given.

> **CAUTION**
>
> If auto-saving is disabled (*Nosave* mode) using the *toggle_save* command, the track is reformatted without preserving the track's data.

Arguments:

- *cyl hd* — Track to reformat

In the following figure, cylinder 300, head 15 is reformatted on an NEC drive (760 cylinders, 19 heads, 59 logical sectors). The track contains one slipped sector.

## Figure dev5130–24, Restoring Track Data

```
(D3;Save)-> format 300 15
        No data has been previously saved. Saving this track
        Data save operation complete

 TRACK HEADERS BEFORE FORMAT:
48 49 50 51 52 53 54 55 56 57 58  0  1  2  3 *B  4  5  6  7  8  9 10 11 12
13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37
38 39 40 41 42 43 44 45 46 47

        About to format cylinder 300  head 15.
        Flawed sectors associated with this track:
            Manufacturer's Defect list: 4
                These flaws will be reapplied to the track since this list is
                not ignored. To ignore manufacturer's defects, you must
                reformat the drive and change format options.
            Grown Defect list:          none
        Format complete.
        Restoring data to track
        Data restore operation complete

(D3;Save)-> track 300 15
 cylinder = 300  head = 15
48 49 50 51 52 53 54 55 56 57 58  0  1  2  3 *B  4  5  6  7  8  9 10 11 12
13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37
45 46 47 48 49 50 51 52 53 54
(D3;Save)->
```

**CAUTION**

If auto-save is enabled and sectors are unslipped by the *format* command, data is restored to the bad sectors which could cause the data to be no longer readable. This is okay since the data is still saved. Slip the bad sectors using the *slip_sectors* command and then use the *restore_track_data* command to restore the track's data again. Also before the reformat, a prompt is displayed to enter which of the grown defects to reslip which allows the reslip to be performed in the same operation.

**NOTE**

If Manufacturer's Defects have been slipped on a track that is to be reformatted, the defects are reslipped automatically after the format. The only way to unslip Manufacturer's Defects is to reformat the whole disk and specify at format setup time (Subtest 300, Format Setup) that the Manufacturer's Defects are to be ignored by using the *format_options* command.

The following is another example where the entire track has been previously mapped to an alternate track because there were two grown defects (sectors 2 and 3) on the track and only one spare sector was available. The alternate track that was used is cylinder 757 head 13.

## Figure dev5130–25, Restoring Data After Testing and Slipping Tracks

```
(D3;Save)-> format 422  1
        No data has been previously saved. Saving this track
        Data save operation complete

  TRACK HEADERS BEFORE FORMAT:
 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13
 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13
 13 13 13 13 13 13 13 13 13 13

        About to format cylinder 422  head  1.
        Flawed sectors associated with this track:
            Manufacturer's Defect list: none
            Grown Defect list:          2 3
            Enter sectors from GD list you want reslipped
              separated by spaces: 2
         Track is currently mapped to cyl 757 hd 13.
         Do you want this alternate to be marked unusable [yn] (n) -> y
         Format complete.
         Restoring data to track
         Data restore operation complete

(D3;Save)-> track 422  1
 cylinder = 422  head =  1
 55 56 57 58  0  1 *B  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19
 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 45 46 47 48 49 50 51
 52 53 54 48 49 50 51 52 53 54
(D3;Save)->
```

When the format is complete, the only sector still bad is sector 2 and the track is no longer mapped to an alternate track.

### Subtest 400, Display Header, Data and ECC

**hd[e_display]** *cyl hd sec* [**to** *cyl hd sec*]

This command displays the sector data and the header and Error Correction Code (ECC). The data is read in raw mode which means the controller does not perform error checking of the data, so an ECC error is not displayed while performing this operation.

Arguments:

- *cyl hd sec* — Starting sector.
- [**to** *cyl hd sec*]— Ending sector, default is starting sector.

> **NOTE**
>
> The sector specified for the *hde_display* command is the physical sector numbered from index. All other commands search all headers on the track until the header is found.

To illustrate how the *hde_display* command works, the track headers displayed by the *track* command and the sector displayed by the *hde_display* command are shown in the following figure:

**Figure dev5130–26, Displaying the Track Headers**

```
(D2;Save)-> track 27 5
 cylinder =  27  head =  5
 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 *S  0
  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25
 26 27 28 29 30 31 32 33 34 35

(D2;Save)-> hde 27 5 0

Phys:  27  5   0  Log:  27  5  36       Header:a81b0524 05240000 ECC:2bc18e26
000  6db66db6 6db66db6 6db66db6 6db66db6  6db66db6 6db66db6 6db66db6 6db66db6
020  6db66db6 6db66db6 6db66db6 6db66db6  6db66db6 6db66db6 6db66db6 6db66db6
040  6db66db6 6db66db6 6db66db6 6db66db6  6db66db6 6db66db6 6db66db6 6db66db6
060  6db66db6 6db66db6 6db66db6 6db66db6  6db66db6 6db66db6 6db66db6 6db66db6
080  6db66db6 6db66db6 6db66db6 6db66db6  6db66db6 6db66db6 6db66db6 6db66db6
0a0  6db66db6 6db66db6 6db66db6 6db66db6  6db66db6 6db66db6 6db66db6 6db66db6
0c0  6db66db6 6db66db6 6db66db6 6db66db6  6db66db6 6db66db6 6db66db6 6db66db6
0e0  6db66db6 6db66db6 6db66db6 6db66db6  6db66db6 6db66db6 6db66db6 6db66db6
100  6db66db6 6db66db6 6db66db6 6db66db6  6db66db6 6db66db6 6db66db6 6db66db6
120  6db66db6 6db66db6 6db66db6 6db66db6  6db66db6 6db66db6 6db66db6 6db66db6
140  6db66db6 6db66db6 6db66db6 6db66db6  6db66db6 6db66db6 6db66db6 6db66db6
160  6db66db6 6db66db6 6db66db6 6db66db6  6db66db6 6db66db6 6db66db6 6db66db6
180  6db66db6 6db66db6 6db66db6 6db66db6  6db66db6 6db66db6 6db66db6 6db66db6
1a0  6db66db6 6db66db6 6db66db6 6db66db6  6db66db6 6db66db6 6db66db6 6db66db6
1c0  6db66db6 6db66db6 6db66db6 6db66db6  6db66db6 6db66db6 6db66db6 6db66db6
1e0  6db66db6 6db66db6 6db66db6 6db66db6  6db66db6 6db66db6 6db66db6 6db66db6

(D2;Save)->
```

The previous figure shows that the physical sector 0 corresponds to logical sector 36 on this track.

The following figure shows the result of an attempt to display header, data, and ECC for track that has been mapped to an alternate track:

## Figure dev5130–27, Displaying a Mapped Track

```
(D2;Save)-> track 280 5
 cylinder = 286  head =  5
 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12
 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12
 12 12 12 12 12 12 12 12 12 12

(D2;Save)-> hde 286 5 0

Phys: 286  5   0  Log: TRK MAPPED to  757 12     Header:d11e0512 02f50000
```

The following figure shows the result of an attempt to display header, data, and ECC of an alternate track. Specifically, the figure shows the header display (*hde*) of cylinder 757 head 12 physical sector 0 from the previous figure where cylinder 286 head 5 has been mapped to cylinder 757 head 12 physical sector 0.

## Figure dev5130–28, Displaying an Alternate Track

```
(D2;Save)-> track 757 12
 cylinder = 757  head = 12
 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 *S  0
  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25
 26 27 28 29 30 31 32 33 34 35

(D2;Save)-> hde 757 12 0

Phys: 757 12   0  Log: 286  5  36       Header:a91e0524 05240000
                      This appears to be an alternate track
```

### Subtest 400, Display List of Commands and Arguments

**h[elp]**

This command lists the commands and their arguments.

### Subtest 400, List Flaws

**l[ist] [m|g|a]**

This command lists flaws which have been fixed on the drive. The types of flaws to be listed can be selected as follows:

- [**m**] — List Manufacturer's Defects entries only
- [**g**] — List Grown Defects entries only
- [**a**] — List Manufacturer's Defects and Grown Defects entries, default value

## Figure dev5130–29, Example of the List Command

```
format setup (D1;Sector)-> list

Summary of flaws on drive 1 serial number 003013
 CYL HD SEC   BCAI   LEN TYPE  ERR SRC | CYL HD SEC   BCAI   LEN TYPE  ERR  SRC
  17  7  23  35032  4825       NONE USR | 237  0   1   4816    17  MAP  HNF  MAP
 237  0   0   4802   105  MAP HARD MAP |
```

The listing above is double-columned and shows the location of all known flaws for this drive. The fields in each entry have the following meanings:

| | |
|---|---|
| CYL HD SEC | The cylinder, head and sector impacted by the flaw |
| BCAI | Byte count after index to the start of the flaw |
| LEN | Length of the flaw in bits |
| TYPE | MAP or blank. If MAP, the entire track has been mapped to an alternate track |
| ERR | The actual error that was detected during pattern test: |

## Table dev5130–13, Pattern Test Flaws

| POSSIBLE VALUES | ERROR CODE | MEANING |
|---|---|---|
| CSUM | 0x19 | Header checksum error |
| HPAD | 0x22 | Invalid header pad |
| SOFT | 0x13 | Soft ECC error |
| HARD | 0x23 | Hard ECC error |
| HNF | 0x29 | Sector not found |
| SYNC | 0x2b | Invalid sync in data field |
| JHDR | 0x6a | Unrecognized header field |
| MHDR | 0x6b | Mapped header error |
| MISC | | Any other error |
| NONE | | No error detected during pattern test |

SRC Where error originated from:

**Table dev5130–14, Sources of Flaws**

| POSSIBLE VALUES | MEANING |
|---|---|
| MAP | Came from original Manufacturer's Defect Map |
| PGM | Discovered during pattern test (Subtest 301) |
| USR | User input this flaw |

**Subtest 400, Pattern Test Range of Sectors**

**p[attern test]** [*nnn* **times**] *cyl hd sec* [**to** *cyl hd sec*] [**with** *pat1* [*pat2* ...]]

This command allows the pattern test of selected sectors repetitively. It is useful to check part or all of a system portion of a disk (all cylinders before the diagnostic cylinder and relocated sectors). If the test is constrained to one track, the data is automatically saved and restored when the test is completed. If more than one track is selected, a warning prompt is displayed and the operation can be terminated at this point to prevent permanent overwrite of data. The automatic save can be disabled with the *toggle_save* command. Up to 15 patterns can be specified. In the list of arguments, [**to** *cyl hd sec*] may be substituted with [**to** *end*].

Arguments:

- [*nnn* **times**]— Repetition count. Possible responses include:
  - **1** — Lower bound value, default value.
  - **2,147,483,647** — Upper bound value.
- *cyl hd sec* — Starting sector.
- [**to** *cyl hd sec*]— Ending sector, default is starting sector.
- [**with** *pat1* [*pat2* ...]] — Up to 15 1-8 digit hex patterns can be specified, default is the patterns specified in Subtest 301, Format and Pattern Test or the patterns in the file *dev5130.patt* if this file exists.

The following figure shows the use of the pattern test command:

**Figure dev5130–30, Example of Pattern Test Command**

```
(D3;Save)-> patt 10 times 0 0 0 to 1 0 0 with 00000000 ffffffff
     Patterns used: 00000000 ffffffff
```

### Subtest 400, Enable or Disable Write Protection

**pr[otection] [on | off]**

This command enables or disables writes to protected areas.

> **CAUTION**
>
> This command should be used with caution. If protection is turned off, writes can be made anywhere on the disk and can destroy unrecoverable information. If protection is on, writes are only allowed to the alternate track area or the used tracks of the defect cylinder.

On ESDI drives, write are also not allowed to the innermost cylinder which contains the original Manufacturer's Defect_map.

### Subtest 400, Exit Interactive Test

**q[uit]**

This command is used to exit the interactive test.

### Subtest 400, Restore Previously Saved Track Data

**r[estore_track_data]**

This command can restore data from the general buffer to the track it was originally read from. The *save_track_data* command stores data into the general buffer from a selected track. In addition, the *format, data_verify,* and *pattern_test* commands save into the general buffer before their operation and automatically restore the data afterward. The data remains in the buffer so it can be restored again at a later time if, for instance, the first restore is bad.

If data is saved and is not restored and then an attempt is made to *quit* the interactive test, a message is displayed indicating that there is saved data and a prompt is displayed that allows reentering or exiting the interactive test.

The following figure shows an example of using the restore command:

**Figure dev5130–31, Restoring Track Data**

```
(D3;Save)-> restore
        Data has been saved for drive 3: cylinder 273  head 17
        Do you really want to restore this data? [yn] -> y
        Data restore operation complete
(D3;Save)->
```

If data is not restored, the operation is terminated and the interactive prompt is displayed.

If the current drive has been changed from drive 3 to drive 9 since the last save operation and the restore command is used, the following is displayed:

**Figure dev5130–32, Changing Drives After a Save Operation**

```
(D9;Save)-> restore
        Data has been saved for drive 3: cylinder 273  head 17
        However, drive 9 is currently selected.
        Use 'd[rive_select]' to change drives and try again.
(D9;Save)->
```

If data has not been saved before attempting to restore the data, the following is displayed:

**Figure dev5130–33, Attempting to Restore Data Before Saving**

```
        No data has been previously saved so restore aborted.
(D9;Save)->
```

### Subtest 400, Save One Track of Data

**sa[ve_track_data]** *cyl hd*

This command saves one track of data into the general buffer. The data can be restored with the *restore_track_data* command. Data on relocated sectors is also saved so if a track is reformatted (which removes the relocated sectors) all the track's data can still be restored.

Arguments:

- *cyl hd* — Track to be saved

The following is an example of saving one track:

## Figure dev5130–34, Saving One Track

```
(D4;Nosave)-> save 525 5
        Data has already been saved for drive 4: cyl=601, hd=14
        Are you sure you want to overwrite this data? [yn] -> y
        Data save operation complete
(D4;Nosave)->
```

The prompt in the previous figure specifies *Nosave* which indicates that the command *toggle_save* was previously entered so that saves are not automatic on destructive operations. But, *Nosave* does not effect this command. Normally, *Save* is displayed since disabling automatic save can cause loss of data.

### Subtest 400, Display Data From Range of Sectors

sec[**tor_display**] *cyl hd sec* [**to** *cyl hd sec*]

This command displays the data from one or more consecutive sectors.

Arguments:

- *cyl hd sec* — Starting sector.
- [**to** *cyl hd sec*] — Ending sector, default is starting sector.

The following is an example of using the *Sector_display command*:

## Figure dev5130–35, Displaying Sector Data

```
(D1;Save)-> sector 27 5 55

        Sector  27  5 55:
000  6db66db6 6db66db6 6db66db6 6db66db6  6db66db6 6db66db6 6db66db6 6db66db6
020  6db66db6 6db66db6 6db66db6 6db66db6  6db66db6 6db66db6 6db66db6 6db66db6
040  6db66db6 6db66db6 6db66db6 6db66db6  6db66db6 6db66db6 6db66db6 6db66db6
060  6db66db6 6db66db6 6db66db6 6db66db6  6db66db6 6db66db6 6db66db6 6db66db6
080  6db66db6 6db66db6 6db66db6 6db66db6  6db66db6 6db66db6 6db66db6 6db66db6
0a0  6db66db6 6db66db6 6db66db6 6db66db6  6db66db6 6db66db6 6db66db6 6db66db6
0c0  6db66db6 6db66db6 6db66db6 6db66db6  6db66db6 6db66db6 6db66db6 6db66db6
0e0  6db66db6 6db66db6 6db66db6 6db66db6  6db66db6 6db66db6 6db66db6 6db66db6
100  6db66db6 6db66db6 6db66db6 6db66db6  6db66db6 6db66db6 6db66db6 6db66db6
120  6db66db6 6db66db6 6db66db6 6db66db6  6db66db6 6db66db6 6db66db6 6db66db6
140  6db66db6 6db66db6 6db66db6 6db66db6  6db66db6 6db66db6 6db66db6 6db66db6
160  6db66db6 6db66db6 6db66db6 6db66db6  6db66db6 6db66db6 6db66db6 6db66db6
180  6db66db6 6db66db6 6db66db6 6db66db6  6db66db6 6db66db6 6db66db6 6db66db6
1a0  6db66db6 6db66db6 6db66db6 6db66db6  6db66db6 6db66db6 6db66db6 6db66db6
1c0  6db66db6 6db66db6 6db66db6 6db66db6  6db66db6 6db66db6 6db66db6 6db66db6
1e0  6db66db6 6db66db6 6db66db6 6db66db6  6db66db6 6db66db6 6db66db6 6db66db6

(D1;Save)->
```

The previous example is almost identical to the one for the *hde_display* command. The major difference between the two commands is that with the *sector_display* command, the read is sensitive to data errors. In the raw read mode that *hde_display* uses, no data errors are reported. If a data error does occur while executing the *sector_display* command, then an error message is displayed, and a prompt is displayed to retry, force, or skip the read.

If a retry of the read is specified, the number of read attempts to try before displaying another error message can be specified. From 1 to 2,147,483,647 retries can be specified. Every five retries a seek is alternately performed one cylinder in and back or one cylinder out and back. Any successful read terminates the retries and the data is displayed.

If a force of the error is specified, one more read is performed with a *move bad data* indication to the controller. Whatever is read on this attempt is displayed. An error indication is not displayed if an error occurs on this read.

If a fix of the error is specified, Error Correction Code (ECC) correction is applied if the error is correctable.

If *skip* is specified, the sector is skipped. If there are any more sectors to read in the range specified, then the display of data continues with the next sector.

The following is an example of a read that fails:

**Figure dev5130–36, Failed Read Example**

```
(D1;Save)-> sector 400 22 18 to 400 22 19

dev5130 drv 1 ERROR 1e(cntlr) Soft ECC Error
 cyl:400 hd:22 sec:18

                        1 attempts to read sector failed
                        what to do [# of retries/force/skip/fix] (1) -> 100

dev5130 drv 1 ERROR 1e(cntlr) Soft ECC Error
 cyl:400 hd:22 sec:18

                        100 attempts to read sector failed
                        what to do [# of retries/force/skip/fix] (100) -> skip


        Sector 400 22 19
000  00000000 00000000 00000000 ff300020  1023eee10 99903220 b2d3d4d1 ab222290
020                    ---------------- etc ----------------
```

In the previous figure sector 18 is not displayed since it was skipped. What is displayed is the next sector, sector 19 which was read successfully on the first attempt. If `fix` had been specified for sector 18, it would have been displayed since soft ECC errors were occurring which are correctable.

### Subtest 400, Display or Change Disk Serial Number

**ser[ial_number]** [*number*]

This command displays the drive's serial number. To change the serial number, follow the command with the new serial number.

### Subtest 400, Slip One or More Sectors

**sl[ip_sectors]**

This command is used to slip sectors or map tracks which are generating errors or to enter defect map information to slip sectors. If track data is automatically saved (indicated by the word *Save* in the prompt), an attempt to save and restore all affected tracks is made to minimize data loss unless automatic save is disabled with the *toggle_save* command.

**NOTE**

When entering the defect map for a new disk, a disk that has just been formatted and does not yet contain a file system, the automatic save may be disabled with the *toggle_save* command before entering the *slip_sectors* command. This causes the slip to proceed much faster.

**CAUTION**

If any usable data exists on these tracks, the data is lost.

The *slip_sectors* command starts up a separate command processor which expects bad sectors or defect map information to be entered. There is a help facility which lists the available commands if **h** is entered. The help screen is shown as follows:

**Figure dev5130–37, *Slip_Sectors* Help Screen**

```
SLIP_SECTORS commands:
q[uit]                          - exit from 'slip_sectors'
c[hange_mode]                   - toggles logical sector/defect
                                  map entry mode
cyl hd sec                      - logical sector to be slipped
cyl hd bcai len                 - defect map entry to be slipped
de[lete] cyl hd sec             - delete logical sector entry
de[lete] cyl hd bcai len        - delete defect map entry
e[xecute]                       - slip the sectors now
f[ile] filename                 - get inputs from specified file
h[elp]                          - display this information
l[ist]  [m|g|a|n]               - list inputs made so far
m[ap_track] cyl hd              - map track to an alternate
!unix-command                   - execute unix command
'comment                        - comment; ignored
```

To slip sectors which are generating errors, the normal usage is to enter all sectors which need to be slipped, verify the entries using the *list* command, delete any incorrect entries with the *delete* command, and then begin the slipping with the command *execute*. New sectors can be entered at any time up until the *execute* command is entered. As a safeguard, when the *execute* command is typed, the list of sectors is automatically displayed and a prompt is displayed to confirm the list. By replying **n** to the confirmation, the *slip_sectors* prompt is displayed with all inputs still intact.

### *slip_sectors' change_mode* Command

**c[hange_mode]**

The *change_mode* command is used to toggle between the logical sector and defect map entry mode. The current input mode (logical sector or defect map mode) is revealed in the slip prompt

as shown in the following figure:

**Figure dev5130–38, Changing Entry Modes**

```
slip (D1;Save;Sector)->c
slip (D1;Save;Defect)->
```

***slip_sectors' cyl hd sec* and *cyl hd bcai len* Commands**

*cyl hd sec*

or:

*cyl hd bcai len*

These commands are used to enter the logical sector to be slipped or to enter the defect map entry to be slipped. When a defect is entered, the associated track's headers are listed as shown in the following figure:

**Figure dev5130–39, Entering a Defect**

```
slip (D1;Save;Defect)-> 293 1 572 9510
        Associated logical sectors: 55 56 57
```

The previous figure illustrates that three sectors are affected by the media flaw which starts at byte 572 from index and is 9510 bits long.

If two defects exist that affect the same sector, the logical sector is only entered in the list of flaws once. Subsequent entries of the same sector are ignored and an appropriate warning is displayed.

For instance, if the following two defects exist:

| cylinder | head | bcai | bit-length |
|---|---|---|---|
| 10 | 0 | 70 | 1 |
| 10 | 0 | 80 | 1 |

Both defects fall in sector 0 so when the second defect is entered, it is not slipped again. The slipping of sectors with two defects is shown in the following figure:

## Figure dev5130–40, Slipping Sectors With Two Defects

```
slip (D1;Save;Defect)-> 10 0 70 1
       Associated logical sectors: 0
slip (D1;Save;Defect)-> 10 0 80 1
                     Cyl 10 Hd 0 Sec 0 already exists. Duplicate entry not created.
slip (D1;Save;Defect)-> list

Summary of new flaws on drive 1 serial number 02039
 CYL HD SEC   BCAI   LEN TYPE  ERR SRC | CYL HD SEC   BCAI   LEN TYPE  ERR SRC
  10  0   0     70     1      NONE NEW |
```

**NOTE**

In the current version of *dev5130*, no bytes are considered part of a gap. A flaw in any byte *results* in one or more sectors being slipped.

### *slip_sectors' delete* Command

**de[lete]** *cyl hd sec*

or:

**de[lete]** *cyl hd bcai len*

The *delete* commands listed above are used to delete logical sector entries or to delete defect map entries. The following figure shows how to delete bad inputs. The *list* command is used after the *delete* command to verify that the bad inputs were deleted.

## Figure dev5130–41, Deleting Bad Inputs

```
(D1;Save)-> slip
Type 'h' for help with slip commands
slip (D1;Save;Sector)-> 17 7 23
slip (D1;Save;Sector)-> 17 7 46
slip (D1;Save;Sector)-> d 17 7 46
       Sector  17  7 46 has been removed from the inputs
slip (D1;Save;Sector)-> list

Summary of new flaws on drive 1 serial number 02039
 CYL HD SEC   BCAI   LEN TYPE  ERR SRC | CYL HD SEC   BCAI   LEN TYPE  ERR SRC
  17  7  23                   NONE USR |
```

### *slip_sectors' execute* Command

**e[xecute]**

The *execute* command is used slip the sectors specified. When the slip is executed, the sectors are slipped one track at a time or the track is mapped if not enough spare sectors exist. If automatic save is active, the data for the track is copied to a special buffer (not the general buffer used by *save_track_data*) before the slip and is restored after the slip.

To verify the slip, display the track headers once more using the *track_headers_display* command. All slipped sectors are marked with a **B*. Mapped tracks have the same number for all sectors. This number is the head number of the alternate track. An example of displaying the track headers after slipping sector 255 0 6 is shown in the following figure:

## Figure dev5130–42, Display Slipped Track Headers

```
(D1;Save)-> track 255 0
cylinder = 255  head =  0
 0  1  2  3  4  5 *B  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23
24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48
49 50 51 52 53 54 55 56 57 58
```

If defect input mode is activated and an attempt is made to slip the sector on track 255 head 0 which has already been marked bad in the previous figure, the following figure shows what would be displayed:

## Figure dev5130–43, Slipped Tracks Previously Marked Bad

```
slip (D1;Save;Defect)-> 255 0 3800 1
        Cyl 255 Hd 0 Sec 6 already exists. Duplicate entry not created.
```

### *slip_sectors' file* Command

**f[ile]** *filename*

The file command is used to get inputs from a specified file. To input defects from a file, first create the file as documented in Subtest 300, Format Setup. Then type the command **file** *filename* where *filename* represents the actual filename. The defects in this file can be in byte-count-after-index or logical sector format. The file can also contain track entries which specify tracks to be mapped to alternate tracks.

### *slip_sectors' help* Command

**h[elp]**

This command lists the commands and their arguments.

### *slip_sectors' list* Command

**l[ist] [m|g|a|n]**

This command lists flaws which have been fixed on the drive. The types of flaws to be listed can be selected as follows:

- [m] — List Manufacturer's Defects entries only
- [g] — List Grown Defects entries only
- [a] — List Manufacturer's Defects and Grown Defects entries, default value
- [n] — List defect needs

### *slip_sectors' map_track* Command

**m[ap_track]** *cyl hd*

To map an entire bad track, use the *map_track cyl hd* command. The test selects one of the alternate tracks for use. If data is being saved, then the original track's data is restored to the alternate track.

### *slip_sectors' quit* Command

**q[uit]**

To exit the *slip_sectors* command at any time enter the *quit* command.

### *slip_sectors'* Execute UNIX Command

*!UNIX-command*

This command is used to issue UNIX commands. After the UNIX command is completed, execution is returned to the interactive prompt. UNIX commands can be issued from any prompt throughout the test.

### *slip_sectors'* Enter Comments

*'comment*

This command is used to enter a comment. To enter a comment, type the ' command and a comment. The comment is echoed so it is redirected output. This is useful to document redirected output.

### *slip_sectors* Command Examples

The following figure contains examples of several *slip_sectors* commands.

## Figure dev5130–44, Command Examples

```
slip (D1;Save;Sector)-> 17 7 36
slip (D1;Save;Sector)-> 247 12 4
slip (D1;Save;Sector)-> 256 0 12
slip (D1;Save;Sector)-> d 256 0 12
slip (D1;Save;Sector)-> 255 0 12
slip (D1;Save;Sector)-> list

Summary of new flaws on drive 1 serial number 02039
 CYL HD SEC   BCAI   LEN TYPE  ERR SRC | CYL HD SEC   BCAI   LEN TYPE  ERR SRC
  17  7  23  35032  4825  MAP NONE USR | 247 12   4   2432  4825       NONE USR
  17  7  36   6644  4825  MAP NONE USR | 255  0  12   7904  4825  MAP NONE USR

slip (D1;Save;Sector)-> e

Summary of new flaws on drive 1 serial number 02039
 CYL HD SEC   BCAI   LEN TYPE  ERR SRC | CYL HD SEC   BCAI   LEN TYPE  ERR SRC
  17  7  23                   NONE USR | 247 12   4                    NONE USR
  17  7  36                   NONE USR | 255  0  12                    NONE USR

       Are you sure inputs are ready for slipping? [yn] -> y

Number of spares per track: 1
 cyl hd sec  bcai  len  action          additional information
---- -- --- ----- ----- ------- ---------------------------------------------
  17  7  23 35032  4825 mapped  Alternate track used: cylinder 757 head 13
  17  7  36  6644  4825 mapped  Alternate track used: cylinder 757 head 13
 247 12   4  2432  4825

dev5130 drv 1 ERROR 23(cntlr) Hard ECC error
 cyl:247 hd:12 sec:4

                     1 attempts to read sector failed
                     what to do [# of retries/force/skip/fix] (1) -> fix
Number of spares per track: 1
 cyl hd sec  bcai  len  action          additional information
---- -- --- ----- ----- ------- ---------------------------------------------
 247 12   4  2432  4825 slipped
 255  0  12  7904  4825 mapped  Alternate track used: cylinder 757 head 12

slip (D1;Save;Sector)-> q
```

In the previous figure, the *cyl hd sec, delete, list, execute,* and the *quit* commands are displayed. Sector 247 12 4 is difficult to read (hard ECC error). After the first attempt fails, a request is made to fix the error. The next attempt has either a soft Error Correction Code (ECC) error or no error is reported. If an error is not fixable, specifying *force* results in one more read of the sector with the controller set to move bad data. If *skip* is selected, the remaining slips are still performed. Retries are allowed by entering the number of retries or by pressing RETURN to use the default number of retries. Every five retries a seek is alternately performed one cylinder in and

back or one cylinder out and back.

In the following figure the track headers are displayed again to verify the slips:

## Figure dev5130–45, Displaying the Track Headers

```
(D1;Save)-> track 17 7
cylinder =  17  head =  7
 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13
 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13 13
 13 13 13 13 13 13 13 13 13 13

(D1;Save)-> track 247 12
cylinder = 247  head = 12
  0  1  2  3 *B  4  5  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23
 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48
 49 50 51 52 53 54 55 56 57 58

(D1;Save)-> track 255 0
cylinder = 255  head =  0
 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12
 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12 12
 12 12 12 12 12 12 12 12 12 12
```

In the previous figure Track 17 7 and track 255 0 have been relocated to head 13 and head 12 respectively of the relocation area. The cylinder is not known but can be determined by using the *hde_display* command on any sector of the bad track. Track 247 12 has one bad sector and no spare sector.

In the following figure all flaws are redisplayed to verify the relocation actually occurred:

## Figure dev5130–46, Display All Flaws to Verify Relocation

```
(D1;Save)-> list

BAD BLOCK TABLE FOR ccu/mb/csr/d=3/0/3f0/0

Summary of new flaws on drive 1 serial number 02039
 CYL HD SEC   BCAI   LEN TYPE  ERR SRC | CYL HD SEC   BCAI   LEN TYPE  ERR SRC
  17  7  23  35032  4825  MAP NONE USR | 247 12   4   2432  4825       NONE USR
  17  7  36   6644  4825  MAP NONE USR | 255  0  12   7904  4825  MAP NONE USR

(D1;Save)->
```

If there had already been entries in the table, they would have been displayed in the previous figure also. This list of the flaws actually comes from a copy of the table kept in main memory. However, it does reflect what is on the disk since both the *slip_sectors* and *format* commands rewrite all five copies of the defect lists before returning to the interactive prompt. To reread

them off the disk before listing them, use the *status* command. It rereads all lists before displaying the status of the lists.

One other verification to perform is a *verify_format* on each of the tracks.

### Subtest 400, Display Information About Device

**st[atus]**

This command lists the drive configuration data (all selected drives) and displays which drive is the currently active drive. The *Status* command also displays which track was last saved in the general buffer by using one of the commands: *format*, *save_track_data*, *data_verify*, or *pattern_test*. In addition, a line is displayed which indicates whether the automatic save of track data is enabled (*Save* is also shown in the prompt). If the drive is previously formatted, then the following information is also displayed:

a. The status of all copies of defect lists and track logs on the disk.

b. The revision of the formatter that originally formatted the disk.

c. The revision of the formatter that last reformatted the disk.

d. The date the drive was originally formatted.

e. The date the drive was last reformatted.

f. The serial number of the drive.

This information is from the defect list cylinder which is reread each time the *status* command is entered. The following figure shows the information that is displayed when the *status* command is entered:

## Figure dev5130–47, Displaying Device Information

```
(D1;Save)-> status

                        DRIVE CONFIGURATION DATA
      CCU VME Cntlr Int Unit #   # Phys Log Prev
Drive  #   #   CSR Level #  Cyl Hds Sec Sec Fmtd Name          Serial #
----- --- --- ----- --- --- --- --- --- --- --- ------- ----------------------
   1   3   0  0xc00  1   0  760  19  60  59 yes DKD-206 03991
   2   3   0  0xa00  4   0  760  19  60  59 yes DKD-206 03268
   3   6   1  0xc00  3   0  842  20  46  45 yes DKD-208 04123

Drive 1 is currently selected
Saved Track:  drive 3  cylinder 300 head 12
Save of track data is automatic during destructive commands.
Status of defect lists:
Drive  1 All copies are GOOD
Drive information
        Serial number: 03991
        Original Formatter Version: 3.00 Date: Mon Mar 14 17:33:24 CST 1988
        Latest Disk Update Version: 3.00 Date: Tue Mar 15 10:23:14 CST 1988
        Format options: Manufacturer's defect list was used if available
                        Pattern test was completed using pattern set 2

(D1;Save)->
```

### Subtest 400, Enable and Disable of Automatic Save

**to**[*ggle_save*]

This command switches between enabling and disabling of automatic save. Automatic save occurs during the commands *format, pattern_test, data_verify* and *slip_sectors*. The displayed interactive prompt indicates whether automatic save is enabled (Save) or disabled (Nosave). The *Status* command also displays the current setting of automatic save.

### Subtest 400, Display Track Headers

**tr**[**ack_headers_display**] *cyl hd* [**to** *cyl hd*]

This command displays the sector numbers for each of the sectors on a track. The sector number that is displayed first is located physically just after the index so it will not always be zero. This is because sectors may be skewed and interleaved. For a skew of 5, head 0 logical sector 0 is the first sector after index. On head 1, logical sector 0 is the sixth sector, on head 2, logical sector 0 is the eleventh sector and so on.

Arguments:

- *cyl hd* — Starting sector.
- [**to** *cyl hd*] — Ending sector, default is starting sector.

Each track has one spare sector (unless it is used because one or more sectors were slipped on the track). The spare sector is marked *S. If any sectors have been flagged as bad, they are marked *B.

If the entire track has been remapped to an alternate track, then the head number of the alternate track is displayed. To find out what alternate track is being used, use the *hde_display* command.

The following figure shows the display of sector numbers for sectors and a *S for a spare sector on a track:

## Figure dev5130–48, Displaying Sector Numbers and Spare Sector

```
(D1;Save)-> track 427 2
cylinder = 427  head =  2
50 51 52 53 54 55 56 57 58 *S  0  1  2  3  4  5  6  7  8  9 10 11 12 13 14
15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39
40 41 42 43 44 45 46 47 48 49
(D1;Save)->
```

In the previous figure, sector 0 is actually the eleventh sector on the track since it is head 2 with a skew of 5. Also the spare sector (*S) is just before sector 0.

The following figure shows the display of sector numbers for sectors and a *B for a bad sector on a track:

## Figure dev5130–49, Displaying Sector Numbers and Bad Sector

```
(D1;Save)-> track 208 11
cylinder = 208  head = 11

 5 *B  6  7  8  9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28
29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53
54 55 56 57 58  0  1  2  3  4

(D1;Save)->
```

In the previous figure, one sector has been flagged bad and the spare sector has been used. Also sector zero is skewed almost to the end of the track because it is head 11 with a skew of 5.

### Subtest 400, Verify a Range of Sectors

**v[erify_format]** [*nnn* **times**] *cyl hd sec* [**to** *cyl hd sec*]

This command performs repeated reads on a range of sectors in an attempt to detect read errors. The command is not data destructive. It does not perform a data compare since the data is unknown. As the *verify* is executed, a pass count is incremented on the display to indicate how far the execution has progressed. Any errors are displayed without retry and the reads continue. If any sectors have been relocated, the relocated sectors are read. In the list of arguments, [**to** *cyl hd sec*] may be substituted with [**to end**].

Arguments:

- [*nnn* **times**]— Repetition count. Possible responses include:
  - **1** — Lower bound value, default value
  - **2,147,483,647** — Upper bound value
- *cyl hd sec* — Starting sector
- [**to** *cyl hd sec*]— Ending sector, default is starting sector

### Subtest 400, Execute UNIX Command

!*UNIX-command*

This command is used to issue UNIX commands. After the UNIX command is completed, execution is returned to the interactive prompt. UNIX commands can be issued from any prompt throughout the test.

The following figure shows an example of entering the ! command and the UNIX command ps:

## Figure dev5130–50, Executing UNIX Command From Interactive Test Prompt

```
(D1;Save)-> !ps
   PID TTY TIME CMD
     2 co  0:08 -
   273 co  0:00 dshell
   274 co  0:09 dev5130.t -c 4
   281 co  0:00 sh -c ps?
   282 co  0:02 ps
(D1;Save)->
```

### Subtest 400, Enter Comments

'*comment*

This command is used to enter a comment. To enter a comment, type the ' command and a comment. The comment is echoed so it is redirected output. This is useful to document

redirected output.

# Examples of Typical Usage of This Test

This section is designed for the casual user who has a general knowledge of disk drives and wants to do some basic operations on disks such as formatting, verifying previously formatted disks, and slipping a sector when the sector is reported bad by CONVEX UNIX. The following examples assume the drives are already cabled up, the VIOP is working, and the switches and jumpers on the controllers and drives are set properly.

## Formatting One or More Disks at the Same Time

Up to 12 drives can be formatted at the same time while CONVEX UNIX is not running.

> **WARNING**
>
> CONVEX UNIX cannot be running concurrently with the *dev5130* test.

This example assumes all the drives selected have not been previously formatted. Therefore, the manufacturer's original defect map is read from the drive during Subtest 300, Format Setup. If any of the drives are previously formatted on a CONVEX system with a VME controller, then **y** should be entered to the `Previously formatted` prompt and RETURN should be entered to `Serial number` prompt so the serial number is read from the drive. If an incorrect reply is entered to the `Previously formatted` prompt, the test detects the error in Subtest 300, Format Setup and then the reply can be changed.

> **NOTE**
>
> Previously formatted means that Subtest 300, Format Setup has been completed; not that all of Class 3 subtests have been completed.

The sequence of commands to invoke the formatter are shown in the following figure:

**Figure dev5130–51, Formatter Invocation Sequence**

```
(spu)> sysreset; mminit -s; dshell RETURN

CONVEX DIAGNOSTIC SHELL

: test dev5130 -c 3 RETURN
```

The following figure shows the prompts and responses to format three drives which have never been formatted before.

## Figure dev5130–52, Formatting Three Unformatted Drives

```
                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
 1:  Use defaults except allow destructive writes [y,n]
                                                  (y) -> RETURN

              PERIPHERAL CONFIGURATION DATA
    CCU       Chassis   Type      CSR     Int  Unit  Type
   --------   -------  --------  -----   ---  ----  -------
 1) viop 7       0      DKC-204   0xa00    1     0    DKD-206
 2) viop 7       0      DKC-204   0xc00    4     0    DKD-208
 3) viop 7       1      DKC-203   0xa00    3     0    DKD-214
Enter device 99 to begin user-defined configurations or -1 to end selection
 2:  Device selection [-1,1-3,99]                 (-1) -> 1
 3:  Previously formatted on CONVEX system with Interphase 4200/4201 controller
     [y,n]                                        (n) -> RETURN
 4:  Serial number (5 to 31 characters) []
                          (Default: read from drive) -> 03058
 5:  Device selection [-1,1-3,99]                 (-1) -> 2
 6:  Previously formatted on CONVEX system with Interphase 4200/4201 controller
     [y,n]                                        (n) -> RETURN
 7:  Serial number (5 to 31 characters) []
                          (Default: read from drive) -> 04012
 8:  Device selection [-1,1-3,99]                 (-1) -> 3
 9:  Previously formatted on CONVEX system with Interphase 4200/4201 controller
     [y,n]                                        (n) -> RETURN
10:  Serial number (5 to 31 characters) []
                          (Default: read from drive) -> 14539
11:  Device selection [-1,1-3,99]                 (-1) -> RETURN
12:  Enter OK, or :NN to return to question NN [OK]
                                                  (OK) -> RETURN

     ------- summary of inputs prints here -------

***********************************************
* DATA DESTRUCTIVE OPERATION IS ABOUT TO START.*
* VERIFY WRITE-PROTECT IS SET ON ALL DRIVES    *
* EXCEPT THOSE YOU ARE TESTING/FORMATTING.     *
***********************************************

Then reconfirm that you want to continue [yn] -> y
```

## Figure dev5130–52, Formatting Three Unformatted Drives (continued)

```
        Subtest 300  0:00:00 Prepare for format

Type 'h' for help with commands
format setup (D1;Sector)->continue

                     0:07:50 Reading Manufacturer's Defect Maps
        Preparing cylinder for defect lists on drive(s) 1 2 3
                     0:08:30  passed
        Subtest 301  1:40:00  passed
        Subtest 302  0:00:00  Fix flaws

Summary of all flaws on drive 1 serial number 03658:
 CYL HD SEC  BCAI   LEN TYPE  ERR SRC | CYL HD SEC  BCAI   LEN TYPE  ERR SRC
   ---------- list of all flaws for this drive prints here ----------

Summary of all flaws on drive 2 serial number 04012:
 CYL HD SEC  BCAI   LEN TYPE  ERR SRC | CYL HD SEC  BCAI   LEN TYPE  ERR SRC
   ---------- list of all flaws for this drive prints here ----------

Summary of all flaws on drive 3 serial number 14539:
 CYL HD SEC  BCAI   LEN TYPE  ERR SRC | CYL HD SEC  BCAI   LEN TYPE  ERR SRC
   ---------- list of all flaws for this drive prints here ----------

                     0:03:00  passed
        Subtest 303  0:11:00  passed
        Subtest 304  0:08:00  passed
        Subtest 305  0:00:50  passed
        Subtest 306  0:00:05  passed

************** Test started Wed Mar 16 08:03:52 1988
************** Test ended   Wed Mar 16 10:23:46 1988

test 'dev5130.t' passed
```

Approximate format times for current CONVEX drives are listed in the following table:

## Table dev5130–15, Drive Format Times

| DRIVE | TIME (hr:min) |
|---|---|
| NEC 2352 (1/2-GByte) | 1:30 |
| NEC 2363 (1-GByte) | 2:30 |
| Hitachi 514-38 | 1:20 |

If one NEC 2352 and one NEC 2363 are being formatted, the format of the drives will take 2.5 hours plus 5 minutes (5 minutes for each additional drive after the first drive). The NEC 2352 will not finish in 1.5 hours in this case because each subtest must complete on all drives before the next subtest begins. The NEC 2352 will finish each subtest before the NEC 2363 but must wait for the NEC 2363 to complete the subtest.

Subtest 306, Verify Pattern Test Error Threshold will fail if any flaws are found on a drive during the pattern test which were not in the existing defects lists. This is not normally a significant concern unless several new flaws are occurring. Inspect the list printed during Subtest 302, Fix Flaws and see how many entries are marked PGM instead of USER or MAP.

## Verifying the Format of One or More Drives

When a disk or set of disks appear to have read problems, verify that the problem repeats by reading all the usable parts of each disk. Then cables or controllers or drives can be changed to verify the disks again. To verify the usable parts of each disk execute Subtest 304, Verify System Format. This subtest performs a read-only operation to locate any unfixed media flaws or to detect problems with the hardware.

The sequence of commands to invoke Subtest 304, Verify System Format are shown in the following figure:

**Figure dev5130–53, Verify Format Invocation Sequence**

```
(spu)> sysreset; mminit -s; dshell RETURN

CONVEX DIAGNOSTIC SHELL

: test dev5130 -s 304 RETURN
```

Upon invocation of Subtest 304, the prompts in the following figure are displayed. Reply to the prompts as shown in the following figure:

## Figure dev5130–54, Verify Format 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
 1:  Use defaults except allow destructive writes [y,n]
                                                    (y) -> n
 2:  Are writes to other than the diagnostics cylinder allowed
    [y,n]                                           (n) -> RETURN
 3:  Are writes to the diagnostics cylinder allowed
    [y,n]                                           (n) -> RETURN
 4:  Number of errors allowed per device each subtest
    [0-65535]                                       (0) -> 100
 5:  Number of devices failed after which test aborts
    [0-65535]                                      (12) -> RETURN
 6:  Threshold for new flaws found during pattern test
    [0-65535]                                       (0) -> RETURN
 7:  Verbosity of test [0-63]                       (3) -> RETURN
 8:  If pattern testing, start after last completed pattern
    [y,n]                                           (y) -> RETURN

                PERIPHERAL CONFIGURATION DATA
     CCU    Chassis   Type     CSR    Int  Unit  Type
    ------  -------  --------  -----  ---  ----  -------
 1) viop 7    0      DKC-204   0xa00   1    0    DKD-206
 2) viop 7    0      DKC-204   0xc00   4    0    DKD-208
 3) viop 7    1      DKC-203   0xa00   3    0    DKD-214
Enter device 99 to begin user-defined configurations or -1 to end selection
 9:  Device selection [-1,1-3,99]                  (-1) -> 1
10:  Previously formatted on CONVEX system with Interphase 4200/4201 controller
     [y,n]                                          (n) -> y
11:  Serial number (5 to 31 characters) []
                          (Default: read from drive) -> RETURN
12:  Device selection [-1,1-3,99]                  (-1) -> 2
13:  Previously formatted on CONVEX system with Interphase 4200/4201 controller
     [y,n]                                          (n) -> y
14:  Serial number (5 to 31 characters) []
                          (Default: read from drive) -> RETURN
15:  Device selection [-1,1-3,99]                  (-1) -> 3
16:  Previously formatted on CONVEX system with Interphase 4200/4201 controller
     [y,n]                                          (n) -> y
17:  Serial number (5 to 31 characters) []
                          (Default: read from drive) -> RETURN
18:  Device selection [-1,1-3,99]                  (-1) -> RETURN
19:  Enter OK, or :NN to return to question NN [OK]
                                                   (OK) -> RETURN

     ------- summary of inputs prints here -------
```

**Figure dev5130–54, Verify Format Parameter Menu (continued)**

```
        Subtest 304  0:03:26
dev5130 drv 2 ERROR 13(cntlr) Soft ECC error
  Cyl: 326 Hd: 7 Sect: 12
                     0:05:41
dev5130 drv 3 ERROR 1e(cntlr) Drive faulted
                     0:08:05  passed

************** Test started Wed Mar 16 08:03:52 1988
************** Test ended   Wed Mar 16 10:23:46 1988

test 'dev5130.t' passed
```

In the previous figure, drive 2 appears to have a new defect at cylinder 326 head 7 sector 12. The defect should be verified before slipping the sector. Refer to the next section for information about verifying a defect and slipping a sector. Drive 3 appears to have a more serious problem. The drive faulted in the middle of the subtest. The fault condition is immediately cleared and the operation is retried. The retry succeeded since the drive passed the subtest. This type of error is difficult to isolate. The drive could actually be reporting a fault condition but the problem could also be cabling or a bad controller. Subtest 400, Interactive Test should be used to isolate intermittent problems.

## Slipping a Sector

The following example shows how the *dev5130* disk formatter can be used to map out media flaws so those portions of the disk are not used. However, before attempting to slip any sectors which have been reported bad by CONVEX UNIX, consider the following points:

1. Is the error at the reported cylinder, head and sector repeatable? Media flaws usually repeat. This example shows how to confirm a flaw. When confirming the flaw, if *dev5130* prints one of the following error messages, then the error is slippable. Otherwise, some other problem that is not media-related is causing the error.

   The following table lists the media-related errors (use code to correlate with UNIX-reported error):

## Table dev5130–16, Media-Related Errors

| ERROR CODE | ERROR DESCRIPTION |
|---|---|
| 0x13 | Soft ECC error |
| 0x19 | Header checksum error |
| 0x22 | Invalid header pad |
| 0x23 | Hard ECC error |
| 0x29 | Sector not found |
| 0x2b | Invalid sync in data field |
| 0x31 | Invalid sync in header |
| 0x6a | Unrecognized header field |
| 0x6b | Mapped header error |

2. The Manufacturer's Defect map is used on all drives when they are formatted. In the past, the map has proven to be reliable in identifying media flaws. Therefore, any new flaws indicate possible drive unreliability. If a drive has new defects, the media may be degrading. Slip the first one or two new defects but if new defects continue, a new drive should be installed.

In the following example, CONVEX UNIX has been reporting a soft Error Correction Code (ECC) error at cylinder 356 head 1 sector 19 on drive 1, an NEC 2352 1/2-GByte drive. An inspection of the file */mnt/errlog* on the service processor indicates the error has occurred three times always at the same location. The system has been brought down to the service processor and *dev5130* is about to be executed. The goal is to perform the following:

1. Check the status of the drive to see if it has had maintenance since it was originally formatted.
2. Verify the error is real and is at the same location. Also, see if old errors are located on the same surface near this flaw. For instance, the cylinder number differs by one or two from a previous error, but the head and sector are the same. This condition is an excellent indicator that a real error exists and the media flaw is impacting more than one track on a surface.
3. Slip the bad sector(s).
4. Verify the entire disk to see if any other grown errors occur.

First, invoke the interactive test as shown in the following figure:

## Figure dev5130–55, Interactive Test Invocation Sequence

```
(spu)> sysreset; mminit -s; dshell RETURN
CONVEX DIAGNOSTIC SHELL
: test dev5130 -c 4 RETURN
```

Upon invocation of the Class 4 subtest, the prompts in the following figure are displayed. Reply to the prompts as shown in the following figure:

## Figure dev5130–56, Interactive 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
 1:  Use defaults except allow destructive writes [y,n]
                                                  (y) -> y

              PERIPHERAL CONFIGURATION DATA
    CCU    Chassis   Type      CSR     Int  Unit  Type
   ------  -------  --------  -----   ---  ----  -------
 1) viop 7    0      DKC-204   0xa00    1    0    DKD-206
 2) viop 7    0      DKC-204   0xc00    4    0    DKD-208
 3) viop 7    1      DKC-203   0xa00    3    0    DKD-214
Enter device 99 to begin user-defined configurations or -1 to end selection
 2:  Device selection [-1,1-3,99]                (-1) -> 1
 3:  Previously formatted on CONVEX system with Interphase 4200/4201 controller
     [y,n]                                        (n) -> y
 4:  Serial number (5 to 31 characters) []
                          (Default: read from drive) -> RETURN
 5:  Device selection [-1,1-3,99]                (-1) -> RETURN
 6:  Enter OK, or :NN to return to question NN [OK]
                                                 (OK) -> RETURN

     ------- summary of inputs prints here -------

**********************************************
* DATA DESTRUCTIVE OPERATION IS ABOUT TO START.*
* VERIFY WRITE-PROTECT IS SET ON ALL DRIVES    *
* EXCEPT THOSE YOU ARE TESTING/FORMATTING.     *
**********************************************

Then reconfirm that you want to continue [yn] -> y

        Subtest 400  0:00:00 Interactive test
Reading defect lists from all selected drives...
Drive  1 All copies are GOOD

                      INTERACTIVE TEST MODE
                      FOR SMD AND ESDI DRIVES

Type 'h' for help
(D1;Save)->
```

Once the `(D1;Save)->` prompt is displayed, any interactive command can be entered.

**STEP 1:** First check the status of the drive.

In the following figure, the *status* command is entered to display the status of the drive.

**Figure dev5130–57, Use of Status Command Example**

```
(D1;save)-> status

                        DRIVE CONFIGURATION DATA
      CCU VME Cntlr Int Unit #   # Phys Log Prev
Drive  #   #   CSR Level #  Cyl Hds Sec Sec Fmtd Name          Serial #
----- --- --- ----- --- --- --- --- --- --- --- ------- ----------------------
   1   7   0  0xa00  1   0  760  19  60  59 yes DKD-206 03658

Drive 1 is currently selected
No track data has been saved yet.
Save of track data is automatic during destructive commands.
Status of defect lists:
Drive  1 All copies are GOOD
Drive information
        Serial number: 03658
        Original Formatter Version: 3.00 Date: Mon Mar 14 17:33:24 CST 1988
        Latest Disk Update Version: 3.00 Date: Wed May 11 14:03:14 CST 1988
        Format options: Manufacturer's defect list was used if available
                        Pattern test was completed using pattern set 2

(D1;Save)->
```

The previous figure shows that the latest disk maintenance was performed on May 11th. Check the log to see what disk maintenance was performed.

**STEP 2:** Verify the error is real and is at the same location.

Two steps can be used to verify the error. First, perform reads on the track in question to see if any media-related errors occur. If errors occur, the sector is bad and needs slipping. If errors do not occur, then check the current list of flaws and see if any old flaws are on the same head within one cylinder of the new flaw. If this condition is true, then a media flaw which was detected on another cylinder is marginally affecting this track so the sector should still be slipped.

The following figure shows the two steps to verify the error.

## Figure dev5130–58, Use of Verify and List Commands Example

```
(D1;Save)-> verify 10 times 356 1 19
verify_format: pass # 4
dev5130 drv  1 ERROR 13(cntlr) Soft ECC error
  Cyl: 356 Hd: 1 Sect: 19
7
dev5130 drv  1 ERROR 13(cntlr) Soft ECC error
  Cyl: 356 Hd: 1 Sect: 19
10
(D1;Save)-> list
Summary of all flaws on drive 1 serial number 03658:
 CYL HD SEC  BCAI   LEN TYPE  ERR SRC | CYL HD SEC  BCAI   LEN TYPE  ERR SRC
   ---------- list of all flaws for this drive prints here ----------
```

The *verify* command shows whether the error repeats when performing reads only. The *list* command shows all flaws that have been slipped. Check this list for flaws near the new flaw. If executing either of these commands indicate that the flaw should be slipped, then continue to the next step. Otherwise, use the *data_verify* and *pattern_test* commands which perform writes before reads. The error may only show up when the data is written and then read.

**STEP 3:** Slip the bad sector(s)

To slip cylinder 356 head 1 sector 19, enter the commands and data shown in the following figure:

## Figure dev5130–59, Use of Slip Command Example

```
(D1;Save)-> slip
Type 'h' for help with slip commands
slip (D1;Save;Sector)-> 356 1 19
slip (D1;Save;Sector)-> execute

Summary of new flaws on drive 1 serial number 03658:
 CYL HD SEC  BCAI   LEN TYPE  ERR SRC | CYL HD SEC  BCAI   LEN TYPE  ERR SRC
 356  1  19 15200  4825      NONE NEW |

       Are you sure inputs are ready for slipping? [yn] (y) -> y

 Number of spares per track: 1
 cyl hd sec  bcai  len  action              additional information
---- -- --- ----- ----- ------- ---------------------------------------------
 356  1  19 15200  4825 slipped

slip (D1;Save;Sector)-> quit
(D1;Save)->
```

The sector is now slipped.

If more than one flaw is to be slipped, enter all flaws before entering the *execute* command at the `slip (D1;Save)->` prompt.

**STEP 4:** Verify the entire disk to see if any other grown errors occur.

The *verify_format* command can be used to verify that the entire usable part of the disk is error-free. To verify the entire usable part of the disk, enter the *verify_format* command and data as shown in the following figure:

**Figure dev5130–60, Use of Verify Format Command Example**

```
(D1;Save)-> verify 0 0 0 to end
verify_format: pass # 1
(D1;Save)-> quit
```

Since errors did not occur when the verify was executed, the entire disk is readable. This verify may not detect intermittent errors which only a long-term read of the disk would find, but the verify does perform a quick check for flaws on the disk. If any additional flaws occur during this verify, repeat the above steps to slip the new flaws.

# Manufacturer's Defect List and Grown Defect List

All drives on Interphase 4200 V/SMD controllers have the innermost cylinder reserved for five copies of the Manufacturer's Defect (MD) list and Grown Defects (GD) list. Drives on Interphase 4201 V/ESDI controllers use the next to the innermost cylinder. This is due to the fact that the original Manufacturer's Defect map is recorded on the innermost cylinder of ESDI drives and is not overwritten by this formatter. Unfortunately, SMD drives have part of the original Manufacturer's Defect map written to every track on the drive so there is no way to preserve it.

Since UNIX must not write to the defect data copies, the UNIX disk parameters file */etc/disktab* specifies one less cylinder than the disk actually has.

## Defect Cylinder Organization on Drives

The defect cylinder on a drive has the following organization:

head 0 — 1st copy of defect data
head 1 — 2nd copy
head 2 — 3rd copy
head 3 — 4th copy
head 4 — 5th copy
heads 5-9 — continuations of 1st through 5th copies, respectively (if needed)
heads 10-14 — continuations of 1st through 5th copies, respectively (if needed)
heads 15-19 — continuations of 1st through 5th copies, respectively (if needed)

The MD list starts in sector 0 of the first track as defined above and continues on consecutive tracks. The GD list starts in the next sector following the last sector used by the MD list.

## Format of Manufacturer's Defect List

The Manufacturer's Defect (MD) list has the following format:

```
word 0            - Magic Number - 0x492ab20e
word 1            - Version Number of formatter when Manufacturer's defect
                    list was written. The number 300 in the lower halfword
                    means version 3.00 which is the initial version of this
                    formatter.
                    The upper halfword is set to zero and is unused for now.
word 2            - Date (seconds since Jan 1, 1970)
word 3            - Number of Manufacturer's Defect Entries  (Assume m entries)
word 4-6          - First of Manufacturer's Defect Entries
     :
word 3m+4 - 3m+6 - Last of Manufacturer's Defect Entries
```

## Format of Grown Defect List

The Grown Defect (GD) list starts in the next available sector after the Manufacturer's Defect (MD) list. It is therefore necessary to read the number of entries in the MD list before the GD list can be located.

The GD list has the following format:

```
word 0            - Magic Number - 0x92b41fe7
word 1
    halfword 0    - Control bits (in binary)
                    sssspppp XXXXXXXm

                    where ssss is the pattern set used during pattern test.
                    The pattern sets are defined in the description of
                    Subtest 301, Format and Pattern Test.
                    pppp is the index (1-based) into the
                    pattern set. If pppp is 1111, then no pattern testing
                    has been done on this drive. If pppp is 0000, then
                    pattern testing was completed. The bit labeled 'm'
                    is set if the MD list was ignored. This is an option
                    at format setup (Subtest 300, Format Setup).

    halfword 1    - Version Number of formatter when Grown Defect List
                    was written. Two implied decimal places. I.E.- the
                    number 300 will mean version 3.00.
word 2            - Date (seconds since Jan 1, 1970)
word 3-10         - Serial Number (up to 31 characters)
word 11           - Number of Grown Defect Entries  (Assume n entries)
word 12-14        - First of Grown Defect Entries
     :
word 3n+12 - 3n+14 - Last of Grown Defect Entries
```

## Format of Entries in the Manufacturer's Defect List and Grown Defect List

Each entry in both the Manufacturer's Defect (MD) list and Grown Defect (GD) list have the following format:

| Bytes | Description |
|---|---|
| 0-1 | Cylinder |
| 2 | Head |
| 3 | Sector position (logical based on no slipped sectors or spares) |
| 4 | Encoded Bit field |

11110000 - Unused in *dev5130*

00001111 - Error program detected - definitions of these bits will change depending on the formatter used

XXXX0000 - not attempted yet or none detected
XXXX0001 - soft ECC
XXXX0010 - header checksum
XXXX0011 - invalid header pad
XXXX0100 - hard ECC
XXXX0101 - sector not found
XXXX0110 - invalid data sync
XXXX0111 - invalid header sync
XXXX1000 - unrecognizable header
XXXX1001 - mapped header error
XXXX1010 - this code is unused
XXXX1011 - other error

5-7 Byte Count after Index
8 Error source - Bit field
XXXXXX00 - User input
XXXXXX01 - Program-detected
XXXXXX10 - Manufacturer's Defect Map
XXXX1XXX - Can't map this flaw - map-track command failed
XXX1XXXX - Track is defective
XX1XXXXX - More than 4 flaws on this track
X1XXXXXX - New entry
1XXXXXXX - Ignore this error
9-11 Bit length

# Track Relocation Area Description

The first five sectors of each of the last five tracks in the track relocation area (track map log and alternate track area) are reserved for holding one copy of the track map log.

Each entry in the log corresponds to a track in the alternate track area. The first entry in the log corresponds to the last track - 5 on the innermost cylinder of the track relocation area. The second entry corresponds to the last track - 6 and so on.

The file */mnt/bin/lib/DB_diskfmt* defines the number of cylinders that are set aside for the track map log and alternate track area. At least one cylinder must be dedicated for the track map log and alternate track area. As of version 3.00 of this formatter, the technique for calculating the number of cylinders is 0.005 * total number of cylinders rounded up. If */mnt/bin/lib/DB_diskfmt* defines more cylinders than are necessary to total 636 tracks (the maximum number of entries allowed in the track map log), then only the innermost 636 tracks of the alternate track area are set aside as alternate tracks. So, tracks beyond those defined in the */mnt/bin/lib/DB_diskfmt* file are not used.

> **NOTE**
>
> If any media flaws fall in the alternate track area, the track is mapped to itself.

For example, on the NEC 2363 drive, if cylinder 1021, head 20 is bad, then the 5th word of the track map log is set to cylinder 1021 head 20 so that it is not used by other tracks.

The association of entries with tracks on a 1 GByte NEC 2363 drive which has cylinders 0-1023, heads 0-26, and sectors 0-66 is the following:

| Track Relocation Area | Entry | Cylinder | Head |
|---|---|---|---|
| Track map log | Log copy 1 | 1021 | 26 |
| | Log copy 2 | 1021 | 25 |
| | Log copy 3 | 1021 | 24 |
| | Log copy 4 | 1021 | 23 |
| | Log copy 5 | 1021 | 22 |
| Alternate track area | Alt trk 0 | 1021 | 21 |
| | Alt trk 1 | 1021 | 20 |
| | : | : | : |
| | Alt trk 21 | 1021 | 0 |
| | Alt trk 22 | 1020 | 26 |
| | Alt trk 23 | 1020 | 25 |
| | : | : | : |
| | -------- etc --------- | | |

### Format of the Track Map Log

The track map log has the following format:

```
word 0          - Magic Number - 0xca29e40b
word 1          - Version Number of formatter when written. The number
                  300 in the lower halfword will mean version 3.00.
                  The upper halfword is set to zero and is unused for now.
word 2          - Date (seconds since Jan 1, 1970)
word 3          - Number of Track Map Entries (one word per entry)
word 4          - Entry 0
                  Upper halfword - cylinder of bad track
                  Lower halfword - head of bad track
words 5 - 639   - Remaining 635 entries
```

## Disk Parameters File, *DB_diskfmt* Description

The disk parameters file, */mnt/bin/lib/DB_diskfmt*, contains information about disk drives. This information is required to be able to format or test new drives. Each line in the file contains a number of fields separated by one or more spaces. Comments start with a # in column 1. To add new drives, create a new entry with all the fields defined, and then add the drive to the */ioconfig* file. As of the time of this printing, the *DB_diskfmt* file contains the following information for all CONVEX machines:

## Figure dev5130–61, Contents of the *DB_diskfmt* File

```
# DB_diskfmt - file of disk parameters
# >>>> WARNING - DO NOT USE 'diskfmt' TO FORMAT! It is no longer compatible
# >>>>           with the CONVEX FORMAT! Instead, format MBUS-attached drives
# >>>>           with 'dev4110' and VME-attached drives with 'dev5130'.
# KEY FOR DRIVE NAMES (unformatted capacity is given in parentheses):
#  Name          Description          Name          Description
# DKD-001       Fujitsu Eagle (452MB) DKD-008,208  NEC 2363 (1080MB)
# DKD-002       CDC 9766 (300MB)      DKD-214      Hitachi DK514-38 (356MB)
# DKD-005,206   NEC 2352 (500MB)
#------- XYLOGICS 450/451 SMD CONTROLLER (MBUS) -------
#  a     b    c    d  e  f    g     h     i j k  l  m   n    o  p
DKD-001 2   842  20 46 45 4800 28160 1 0 1  0  0  smd mfm y
DKD-002 0   823  19 32 31 5040 20160 1 0 1  0  0  smd mfm y
DKD-005 0   760  19 60 59 4832 36288 1 0 1  0  0  smd 2-7 y
DKD-008 1  1024  27 68 67 4816 40960 1 0 1  0  0  smd 2-7 y
#------- INTERPHASE 4201 ESDI CONTROLLER (VME) --------
#  a     b    c    d  e  f    g     h     i j k  l  m   n    o  p
DKD-214 0   903  14 51 50 4736 30240 5 5 1  8  8 esdi 2-7 n
#------- INTERPHASE 4200 SMD CONTROLLER (VME) ---------
#  a     b    c    d  e  f    g     h     i j k  l  m   n    o  p
DKD-206 0   760  19 60 59 4832 36288 5 4 1 12 12  smd 2-7 n
DKD-208 0  1024  27 68 67 4816 40960 5 6 1 16 12  smd 2-7 n
# LEGEND:
# a - drive name                Must be DKD-0XX for Multibus and DKD-2XX for
#                               VMEbus
# b - disk type                 For Xylogics controller
# c - # of cylinders
# d - # of heads
# e - # of physical sectors     Number of actual sectors excluding runt
# f - # of logical sectors      Number of physical sectors (e) minus number of
#                               spares
# g - bits per sector           Number of bits between sector pulses
# h - bytes per track           Total number of unformatted bytes per track
# i - skew                      Sector offset from one head to the next
#                               Must be 1 when using Xylogics 450/451 cntlr
# j - # of relocation tracks    .5% of number of cylinders (c). Raise
#                               fractional part to next higher whole number.
#                               Ignored by dev4110 (Multibus formatter)
# k - interleave                sector separation between consecutive sectors
#                               Currently must be 1 for Xylogics 450/451 cntlr
# l - gap 1 size                Number of halfwords in gap before header
#                               (2 bytes per halfword)
#                               Ignored by dev4110 (Multibus formatter)
# m - gap 2 size                Number of halfwords in gap following header
#                               (2 bytes per halfword)
#                               Ignored by dev4110 (Multibus formatter)
# n - drive interface           Used to determine how to read manufacturer's
#                               defect map.  Currently, smd or esdi
#                               Ignored by dev4110 (Multibus formatter)
# o - data encoding scheme      Way data is encoded on the media. Used to
#                               select patterns for pattern test.
#                               Currently mfm, 2-7 or 1-7
# p - Are spares interleaved    For Xylogics, 'y'. For Interphase, 'n'.
```

# Diagnostic Cylinder Description

The diagnostic cylinder is located on the first full cylinder preceding the sector forwarding area. Cylinder 840 is the diagnostic cylinder on a Fujitsu Eagle drive, and cylinder 820 is the diagnostic cylinder on a CDC SMD drive.

Each track of the cylinder is independent of the other tracks since there is a "table of contents" stored at sector 0 of each track This table describes what is stored on the remainder of that track. The format of this table is shown below:

**Figure dev5130–62, Diagnostic Cylinder Table of Contents Format**

```
/* ------------------------------------------------------------------- *
 *              Diagnostic Cylinder Definitions                        *
 * ------------------------------------------------------------------- */
#define NOT_USED        0x00000000        /* position in tbl not used     */
#define NO_HDR          0x00000001        /* this sectors header deleted  */
#define TBL_CONT        0x00000002        /* sector contains tbl of cntnts*/
#define ECC_ERR         0x00000003        /* sector written with bad ecc  */
#define PATTERN         0x00000004        /* sector written with pattern  */

 struct   tbl_cont {
    int   magic_nbr;            /* identifies this as a table of contents */
    int   version;              /* version number of this table */
    int   cyltksc;              /* sector header for this sector*/
    struct {
     int   sec_cont;          /* defines contents of corresponding sector */
     int   pattern;           /* pattern or mask used to verify contents  */
    } sec_tbl[62];
    int   checksum;          /* arithmetic tally of all the above fields */
 };
```

Each track on the cylinder is formatted in such a way that sector 0 is always on the track; sector 0 is written and verified without error when the drive is formatted.

The first field in the table contains the magic number 0x84101500. If this number is not present in the first position of a table, then someone has altered the diagnostic cylinder.

The second field in the table contains the version number. This number is an ordinal number starting with one and increasing as needed. Its purpose is to allow some measure of compatibility of newer diagnostic cylinders with older software. This number is checked if an unknown entry in the sector table (sec_tbl) is encountered. If the version number in the table is greater than the version number of the software, the diagnostic cylinder is assumed to have been formatted with a newer revision of software, and the opcode in the sector table is new to the older software. Therefore, the software is not familiar with the opcode, and the entry in the sec_tbl is ignored.

The next field in the table contains a copy of the sector header (cyltksc) for the table of contents.

The next field in the table contains the sector table (sec_tbl). This table describes what is stored on the rest of the sectors in this track. The sector table consists of 62 entries of 2 int's. Each entry (0 - 61) in this table, corresponds to a sector (0 - 61) on the track. The first position in each entry is the sector contents (sec_cont) field. If the track has more than 62 sectors, the extra

sectors are not used or checked by current software. The following table describes what is currently defined:

**Table dev5130–17, Defined Values for Sector Contents Field**

| FIELD | DESCRIPTION |
|---|---|
| NOT_USED | This sector has not been initialized and should not be checked. |
| NO_HDR | When this sector is read, the controller should return a 0x05 sector not found error. |
| TBL_CONT | This sector contains a copy of the table of contents. |
| ECC_ERROR | This sector contains a copy of the table of contents that has been corrupted to give some form of ECC error. The pattern field contains a mask which may be xor'ed with the magic number field to correct the error. If the mask contains 11 or fewer one bits, the controller should report a soft ECC error when this sector is read. If the mask contains more than 11 one bits, the error reported should be a hard ECC error. |
| PATTERN | This sector has been filled with the data contained in the pattern field. No errors should be encountered when the sector is read. |

The last field in the table of contents is a checksum. This is an arithmetic tally of all the bytes in the table of contents. This value is checked before any of the data in the table of contents is used.

# Error Messages

The following are error messages for the *dev5130* test. The following error format is generated from a test and is not associated with a controller or drive:

```
dev5130 ERROR 56(test)-Unable to get SPU memory
```

The following error format is generated from a test and is associated with a controller but not with a specific drive:

```
dev5130 ccu/vb/csr 7/0/e00: ERROR 94(test) No cntlr interrupt in 5 seconds
```

The last error format shown below is generated from a controller and is associated with a specific drive:

```
dev5130 drv  1 ERROR 13(cntlr) Soft ECC error
 Cyl: 356 Hd: 1 Sect: 19
```

The following are possible errors:

- 0x10(cntlr) Disk not ready (or, if a DKD-214, "Write Protected")

- 0x12(cntlr) Seek error
- 0x13(cntlr) Soft ECC error
- 0x14(cntlr) Invalid Command Code
- 0x15(cntlr) Illegal fetch and execute
- 0x16(cntlr) Invalid sector in command
- 0x17(cntlr) Illegal memory type
- 0x18(cntlr) Bus timeout
- 0x19(cntlr) Header checksum error
- 0x1a(cntlr) Disk write protected
- 0x1b(cntlr) Unit not selected (or, "Powered Off")
- 0x1c(cntlr) Seek error timeout
- 0x1d(cntlr) Fault timeout
- 0x1e(cntlr) Drive faulted
- 0x1f(cntlr) Ready timeout
- 0x20(cntlr) End of medium
- 0x21(cntlr) Translation fault
- 0x22(cntlr) Invalid header pad
- 0x23(cntlr) Hard ECC error
- 0x24(cntlr) Translation error, cyl
- 0x25(cntlr) Translation error, head
- 0x26(cntlr) Translation error, sect
- 0x27(cntlr) Data overrun
- 0x28(cntlr) No index pulse on format
- 0x29(cntlr) Sector not found
- 0x2a(cntlr) ID field error-wrong head
- 0x2b(cntlr) Invalid sync in data field
- 0x2c(cntlr) No valid header found
- 0x2d(cntlr) Seek timeout error
- 0x2e(cntlr) Busy timeout
- 0x2f(cntlr) Not on cylinder
- 0x30(cntlr) RTZ timeout
- 0x31(cntlr) Invalid sync in header
- 0x3f(cntlr) No heads specified
- 0x40(cntlr) Unit not initialized
- 0x41(cntlr) Not used
- 0x42(cntlr) Gap specification error
- 0x42(test) No consecutive sectors found
- 0x43(test) Flaw log is full
- 0x47(test) Bad parm to calc_cyl

- 0x48(test) Bad parm to calc_hd
- 0x49(test) Bad parm to calc_sec
- 0x4a(test) IOP print utility memory error
- 0x4b(cntlr) Seek error
- 0x4b(test) Device FAILED
- 0x4c(test) Input parms don't allow write
- 0x4e(test) Diag. trk has all bad headers
- 0x4f(test) Diag. cylinder bad
- 0x50(cntlr) Sectors/track error
- 0x50(test) Bad data compare on diag. cyl
- 0x51(cntlr) Bytes/sector spec error
- 0x51(test) No err. Hdr-not-found expected
- 0x52(cntlr) Interleave spec factor
- 0x52(test) Soft ECC error did not occur
- 0x53(cntlr) Invalid head address
- 0x53(test) Hard ECC error did not occur
- 0x54(cntlr) Invalid cylinder addr
- 0x54(test) Data compare err on diag cyl
- 0x55(test) Setup failed - aborting test
- 0x56(test) Unable to get SPU memory
- 0x57(test) Track map log is full
- 0x58(test) No defect lists could be read
- 0x59(test) Can't read Manufacturer's defects
- 0x5a(test) New flaws exceeded user limit
- 0x5b(test) No copies successfully written
- 0x5c(test) Less than 3 good copies of defect lists
- 0x5d(cntlr) Invalid DMA xfer count
- 0x5f(test) Hardware is suspected to be bad
- 0x60(cntlr) IOPB failed
- 0x60(test) Drive is previously formatted
- 0x61(cntlr) DMA failed
- 0x61(test) Drive is not previously formatted
- 0x62(cntlr) Illegal VME address
- 0x62(test) Unable to read track headers
- 0x63(test) Unable to access main memory
- 0x64(test) Mapped track has invalid headers
- 0x65(test) Alternate track has invalid headers
- 0x66(test) Incorrect alloc of pattern space
- 0x67(test) Message received was not handshake

- 0x68(test) Incorrect handshake message format
- 0x69(test) Defect list to long to fit on disk
- 0x6a(cntlr) Unrecognized header field
- 0x6a(test) Prev. formatted drive appears unformatted
- 0x6b(cntlr) Mapped header error
- 0x6b(test) User terminated drive
- 0x6c(test) Controller firmware is out of date
- 0x6d(test) Bad data written by Format with data
- 0x6f(cntlr) No spare sector enabled
- 0x77(cntlr) Command aborted
- 0x78(cntlr) ACFAIL detected
- 0x80(test) Data compare error
- 0x81(test) Seek requested with a seek count = 0
- 0x85(test) Desired IOPB not in active chain
- 0x86(test) Message received on wrong subqueue
- 0x87(test) Wrong subqueue active and locked
- 0x88(test) Wrong subqueue active and MBS error
- 0x89(test) Wrong subqueue active, error invalid
- 0x8a(test) river executed for no reason
- 0x8c(test) ntlr busy after hard error
- 0x92(test) EGOS routine wndw_alloc returned 0
- 0x93(test) Task init attempt-error pending
- 0x94(test) No cntlr interrupt in 5 seconds
- 0x96(test) Write/Read error in controller regs
- 0x9a(test) Finished task but int. state wrong
- 0x9b(test) Controller is not present
- 0x9c(test) Sector interleave or skew error
- 0x9d(test) Bad cylinder address after seek
- 0xa0(test) Expected error did not occur
- 0xa2(test) No free IOPBs for retry
- 0xa3(test) Controller didn't finish IOPB
- 0xa4(test) Unknown error reported by cntlr
- 0xa5(test) Internal Defect log is full
- 0xa6(test) Controller doesn't respond
- 0xa7(test) Track doesn't contain flaw data
- 0xa8(test) Cntlr reporting board not ok
- 0xa9(test) Software error-Invalid verify
- 0xaa(test) Cntlr GO/BUSY set after diag cmd
- 0xab(test) Header contains bad cyl number

- 0xac(test) Header contains bad head number
- 0xad(test) Attempt to configure cntlr failed
- 0xae(test) Unexpected value in cntlr CSR
- 0xaf(test) Bad cyl, hd, or sec in IOPB
- 0xb0(test) NI with more than 1 spare
- 0xc0(cntlr) Both bits set (from appl. note)
- 0xc1(cntlr) MSE without init long (from appl. note)
- 0xff(cntlr) Command not implemented

## Special Error Messages for *dev5130*

In some circumstances, a certain set of error messages are displayed if the peripheral is not powered on, is write protected, or is cabled incorrectly. The test gives the user several options to proceed with, depending on the circumstances. The following examples illustrate these type of error messages and user options that are presented. The first example illustrates the messages produced when the peripheral is not ready (ie., power is not on or a cable is not connected). The second example illustrates the messages produced when the peripheral is write protected and an operation attempts a write.

**NOTE**

In the following examples, all **boldface** entries are user entered information.

## Figure dev5130–63, Drive Not Ready Error Example

```
(D1;Save)-> alternate 0 to 1000
Beginning alternate seeks (1 times)...
dev5130 drv 1 1b(cntlr) Unit not selected
  Cyl:   0 Hd: 0 # Seeks:2
  IOPB: opt  err  cyl hd/s  cnt bufh buf1 bmem inr  dma ptrh ptr1 imem skew
       8102 821b 0000 0000 0001 0001 2000 033d 0100 1000 0000 0000 0000 0000
Attempt to fix the problem. You can issue UNIX commands if it helps.
Then enter on of the following:
        <CR> to retry the operation
        'nofix' if unable to fix the problem
UNIX command, <CR> for retry, or nofix: RETURN

dev5130 drv 1 1b(cntlr) Unit not selected
  Cyl:   0 Hd: 0 Sect:  0 # Seeks:2
  IOPB: opt  err  cyl hd/s  cnt bufh buf1 bmem intr  dma ptrh ptr1 imem skew
       8102 821b 0000 0000 0001 0001 2000 033d 0100 1000 0000 0000 0000 0000
Attempt to fix the problem. You can issue UNIX commands if ti helps.
Then enter on of the following:
        <CR> to retry the operation
        'nofix' if unable to fix the problem
UNIX command, <CR> for retry, or nofix: nofix

(D1;Save)-> alternate 20 times 0 to 1000
Beginning alternate seeks (20 times)....
dev5130 drv 1 1b(cntlr) Unit not selected
  Cyl:   0 Hd: 0 # Seeks:21
  IOPB: opt  err cyl  hd/s  cnt bufh buf1 bmem inr   dma ptrh ptr1 imem skew
       8a02 821b 0000 0000 0001 0000 0000 033d 0100 1000 0000 0000 0000 0000
```

## Figure dev5130–64, Write Protect Error Example

```
(d1;Save)-> pattern_test 1022 10 30 with 0
    Patterns used:
     00000000
       No data has been previously saved. Savin this track
       Data save operation complete
 pass #1
dev5130 drv 1 1a(cntlr) Disk write protected
  Cyl:1022 Hd:10 Sect: 30 # Seeks:2
  IOPB: opt  err  cyl hd/s  cnt bufh buf1 bmem intr  dma ptrh ptr1 imem skew
       8202 821a 03fe 0a1e 0001 0005 2000 033d 0100 1000 0000 0000 0000 0000
Attempt to fix the problem. You can issue UNIX commands if it helps.
Then enter one of the following:
        <CR> to retry the operation
        'nofix' if unable to fix the problem
UNIX command, <CR> for retry, or nofix: nofix
        Restoring data to track

dev5130 drv 1 1a(cntlr) Disk writ protected
  Cyl:1022 Hd:10 Sect: 0 # Seeks:2
  IOPB: opt  err  cyl hd/s  cnt bufh buf1 bmem intr  dma ptrh ptr1 imem skew
       8202 821a 03fe 0a00 0001 0005 2000 033d 0100 1000 0000 0000 0000 0000
                        1 attempts to write sector failed
                        what to do [# of retries/force/skip] (1) ->skip
```

## Figure dev5130–65, Power Off Error Message

```
(D1;Save)-> verify_format 0 0 0
verify_format: pass #1
dev5130 drv 1 10(cntlr) Disk no ready
  Cyl:   0 Hd: 0 Sect:  0 # Seeks:2
IOPB: opt  err  cyl hd/s  cnt bufh buf1 bmem inr  dma ptrh ptr1 imem skew
     8102 8210 0000 0000 0001 0001 2000 033d 0100 1000 0000 0000 0000 0000
Attempt to fix the problem. You can issue UNIX commands if it helps.
Then enter on of the following:
        <CR> to retry the operation
        'nofix' if unable to fix the problem
UNIX command, <CR> for retry, or nofix: nofix
```

# dev5200

# VMEbus STC Tape Unit Controller Test

## Overview

The *dev5200* test verifies the functionality of CONVEX VMEbus STC tape controllers.

## Prerequisites and Required Equipment

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

**Table dev5200–1, Hardware Requirements**

| C1, C120 | C200 Series |
|---|---|
| MCU | Memory System[1] |
| MAU | CPX |
| SPU | SP2 |
| VIOP | VIOP |
| VBCU | PIA |
| VBTC[2] | VBCU |
| Tape Drive | VBTC[2] |
| | Tape Drive |

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

[2] VBTC represents VMEbus Tape Controller

## Test Invocation

The *dev5200* 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 *dev5200* 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 dev5200–1, Test Invocation Sequence

```
(spu)> cd /mnt/test RETURN
(spu)> sysreset RETURN
(spu)> mminit -s RETURN
(spu)> dshell RETURN
: test dev5200 [-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 dev5200** executes all *dev5200* 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 powerup. 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 if 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 dev5200–2, Alternate Test Invocation Sequence

```
(spu)> cd /mnt/test RETURN
(spu)> initall RETURN
(spu)> dshell RETURN
: test dev5200 [-c [class number(s)]] [-s [subtest number(s)]] [+> filename] RETURN
```

## 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 dev5200–2, Getting Help During Test Parameter Entry

| Character | Description |
|---|---|
| ? | Displays this help menu |
| h | Provides help for a specific prompt |
| i | Displays the */ioconfig* file |

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

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

## Figure dev5200–3, 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

                PERIPHERAL CONFIGURATION DATA
      CCU       Chassis Type   CSR     Int     Unit    Type
      ------    ------- ------- -----   ---     ----    -------
 1)   VIOP 1       1     MTC-201 0x3fc0  7        0     MTD-201

Device 0 = user defined configuration

 1: Device Selection [0,1][1]                          (1) ->
 2: VIOP [0-3][2]                                       (1) ->
 3: VMEbus Chassis Number [0-1]                            (1) ->
 4: Controller VMEbus Address [0x0-0xffff]          (0x3fc0) ->
 5: Controller VMEbus Interrupt level [0-7]              (7) ->
 6: Tape Unit [0-3]                                 (0) ->

        Tape Unit Types:
        0 -> STC 2921
        1 -> STC 2922
        2 -> STC 1968
        3 -> FUJITSU M2436

 7: Tape Unit Type [0-3]                                (0) ->
 8: Use Defaults for remaining parms [y,n]              (y) ->
 9: Select VBTC VMEbus address modifier [0x39,0x3d]
                                                     (0x39) ->
10: Media Error Retry Count in Subtests n50-n53 [0-15]
                                                        (5) ->
11: Display Each Media Error [y,n]                      (n) ->
12: Run Subtests n50-n53 to EOT [y,n]                   (n) ->
13: Fixed Length Block Size, Subtests n10-n12,n30,n50,n51
    [18-131071]                                     (8192) ->
14: Run EOT Subtest n25 [y,n]                           (y) ->
15: Allow Manual Intervention [y,n]                     (y) ->
16: Enter OK, or :NN to return to question NN [OK]
                                                       (OK) ->
```

[1] The options for this prompt change depending on how many tape units are in the system configuration.

[2] The options for this prompt change depending on the machine architecture. For C1 and C120 machines, the available selection ranges from 3-7. For C200 Series machines, the available selection ranges from 0-3.

> **NOTE**
>
> In prompts 10, 12, 13, and 14 in the previous figure, *n* varies from 3 to 5 depending on which classes are executed. Class 3 subtests execute at 800 bpi, Class 4 subtests execute at 1600 bpi, and Class 5 subtests execute at 6250 bpi. Refer to class and subtest descriptions for more information.

At any time during the test parameter sequence, several options are available as denoted at the top of the Test Parmeter 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

**?** — Request help for the current prompt (if available)

**^** — Return to the previous prompt

## Prompt Explanations

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

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

This prompt selects a device for testing from the system configuration file (*/ioconfig*). If **0** is entered, then user-defined configuration prompts (2-5 in the previous figure) are displayed to determine the configuration to be tested. If a specific device is to be selected, the number of the device is entered (**1**). At that point, the additional configuration prompts (2-5 in the previous figure) are not displayed.

```
VIOP [0-3]                                                (1) ->
```

Enter the CCU slot number for the desired VIOP.

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

Enter the number of the chassis to be tested. (The drive is attached to a controller which is in a VMEbus chassis.)

```
Controller VMEbus Address [0x0-0xffff]
                                                    (0x3fc0) ->
```

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

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

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

```
Tape Unit [0-3]                                    (0) ->
```

Enter the unit number of the tape unit to be tested.

```
Tape Unit Type [0-3]                               (0) ->
```

Enter the number of the type of tape unit to be tested.

```
Use Defaults for remaining parms [y,n]             (y) ->
```

This prompt allows the automatic selection of the default parameters for the remaining prompts.

```
Select VBTC VMEbus address modifier [0x39,0x3d]          (0x39) ->
```

This prompt allows the specification of the address modifier. The following list defines the available options:

- 0x39—Standard Non-Privileged Data Access (16 bits)
- 0x3d—Standard Supervisory Data Access (16 bits)

```
Media Error Retry Count in Subtests n50-n53 [0-15]
                                                   (5) ->
```

Enter the number of retries to be performed when a media error is detected during Subtests n50–n53.

```
Display Each Media Error [y,n]                     (n) ->
```

Enter **n** or RETURN so media errors and retries are not displayed during all subtests; otherwise, enter **y** to display all media errors and retries during all subtests.

```
Run Subtests n50-n53 to EOT [y,n]                    (n) ->
```

Enter **y** to write as many files of data as are required to the end of the tape during Subtests n50–n53.

```
Fixed Length Block Size, Subtests n10-n12,n30,n50,n51
[18-131071]                                        (8192) ->
```

Enter (RETURN) to select 512 bytes as the block size of records written to the tape during Subtests n10–n12, n30, n50, n51; otherwise, enter the block size (in bytes) of records written to the tape.

```
Run EOT Subtest n25 [y,n]                            (n) ->
```

Enter **n** or (RETURN) to skip Subtest n25; otherwise, enter **y** to allow execution of Subtest n25, End-of-Tape Sensing.

```
Allow Manual Intervention [y,n]                      (y) ->
```

Enter **n** to skip Subtests 104, 105, 200–202; otherwise, enter **y** to allow execution of Subtests 104, 105, 200–202.

```
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 dev5200–4, Sample Test Parameter Summary

```
                TEST PARAMETER SUMMARY

Device Selection                                    : 0
VIOP                                                : 1
VMEbus Chassis Number                             : 0
Controller VMEbus Address                         : 0xc0
Controller VMEbus Interrupt level                   : 4
Tape Unit                                           : 0
Tape Unit Type                                      : 0 (STC 2921)
Use Defaults for remaining parms                    : n
Select VBTC VMEbus address modifier                 : 0x39
Media Error Retry Count in Subtests n50-n53         : 5
Display Each Media Error                            : n
Run Subtest n50-n53 to EOT                          : n
Fixed Length Block Size, Subtests n10-n12,n30,n50,n51
                                                    : 8192
Run EOT Subtest n25                                 : y
Allow Manual Intervention                           : y
Enter OK, or :NN to return to question NN         : OK
```

# Hardware Initialization Sequence

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

- A sysreset is performed
- Main memory is allocated for the test
- SPU windows to main memory are initialized
- SPU local test variables are initialized
- The VIOP is booted and loaded
- A driver on the VIOP is started
- VIOP local test variables are initialized

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

# Class Descriptions

The *dev5200* test contains the five classes of subtests listed in the following table:

**Table dev5200–3, *dev5200* Test Classes**

| CLASS | DESCRIPTION |
|---|---|
| 1 | Controller loopback tests |
| 2 | Operator controls tests |
| 3 | Functional tests—800 bpi |
| 4 | Functional tests—1600 bpi |
| 5 | Functional tests—6250 bpi |
| 6 | Speed select tests |

All of the subtests contained in *dev5200* are loopable under the *dshell*.

## Class 1 Subtests

Class 1 subtests verify the basic functionality of the controller. The subtests verify board reset and loopback capabilities. By entering **n** (the default) for the `Allow Manual Intervention` test parameter prompt, Subtests 104 and 105, which require manual intervention are not executed.

**Table dev5200–4, Class 1 Subtests**

| SUBTEST | DESCRIPTION | TIME (min:sec) |
|---|---|---|
| 100 | VBTC Reset Capability | 00:01 |
| 101 | VBTC Interrupt Loopback | 00:05 |
| 102 | VBTC Data Loopback | 00:25 |
| 103 | VBTC Memory Access | 04:30 |
| 104 | VBTC Command Loopback | 00:10 |
| 105 | VBTC Chain Command Loopback | 00:10 |

The execution times in the previous table are approximate times for testing a STC 2920 drive at 1600 BPI. Other drives and densities will affect these figures. Specifying a block size larger than the default changes the runtimes for these subtests. All times include rewind time from the tape position at the end of the last subtest.

The *dev5200* test does not test the drive commands: *DMS*, *SNS*, and *LWR*. In addition, it does not test the following controller-error handling capabilities:

- Pending command abort
- VMEbus timeout

Also, certain drive capabilities are not tested. This includes all command reject and abort scenarios.

### Subtest 100, VBTC Reset Capability

Subtest 100 verifies that the controller resets properly by checking the status register for the correct value.

### Subtest 101, VBTC Interrupt Loopback

Subtest 101 verifies that the board interrupt capability can be turned off and on.

### Subtest 102, VBTC Data Loopback

Subtest 102 tests the direct memory access First-In-First-Out (FIFO) memory on the board. It pattern checks all RAM's in the FIFO and tests in/out capability, assuming the FIFO operates like a circular buffer with pointers.

### Subtest 103, VBTC Memory Access

Subtest 103 checks the capability of the board to access all allowed pages in main memory via the DMA and associated buffers.

### Subtest 104, VBTC Command Loopback

Subtest 104 checks the capability of the board to properly register command status for the executing and pending states. This subtest does not execute when **n** (the default) is entered selected for the test parameter prompt `Allow Manual Intervention`.

### Subtest 105, VBTC Chain Command Loopback

Subtest 105 checks the capability of the board to properly register command status while executing chaining commands. This subtest does not execute when **n** (the default) is entered for the test parameter prompt `Allow Manual Intervention`.

## Class 2 Subtests

Class 2 subtests verify operator controls, tape write protect capability, and unload reel capability. All subtests in this class are skipped if **n** (the default) is entered for the test parameter prompt `Allow Manual Intervention`.

**Table dev5200–5, Class 2 Subtests**

| SUBTEST | DESCRIPTION | TIME (min:sec) |
|---|---|---|
| 200 | VBTC Drive Online | Manual intervention time |
| 201 | VBTC Drive Write Protect | Manual intervention time |
| 202 | VBTC Drive Unload | Manual intervention time |

### Subtest 200, VBTC Drive Online

Subtest 200 verifies that drive online status is properly reflected to the software as the operator manipulates the drive controls. This subtest does not execute when **n** (the default) is entered for the test parameter prompt `Allow Manual Intervention`.

### Subtest 201, VBTC Drive Write Protect

Subtest 201 verifies that file protection is properly reflected to the software as the operator removes and reinserts the write ring. This subtest does not execute when **n** (the default) is entered for the test parameter prompt `Allow Manual Intervention`.

### Subtest 202, VBTC Drive Unload

Subtest 202 tests the drive unload operation. This subtest does not execute when **n** (the default) is entered for the test parameter prompt `Allow Manual Intervention`.

## Class 3, 4, and 5 Subtests

Classes 3, 4, and 5 contain subtests to verify correct operation of the controller and drive for a wide selection of commands and data transfer scenarios, including chaining mode. These subtest are in the range of 300-599 (n00-n99). Subtest numbers in these three classes contain a prefix of *n* (as the hundreds digit), which is used to select tape density. The 300 series subtests are for 800 bpi tapes, 400 series for 1600 bpi, and 500 series for 6250 bpi.

**Table dev5200–6, Class 3, 4, and 5 Subtests**

| SUBTEST | DESCRIPTION |
|---|---|
| n00[1] | VBTC Write Tape Mark |
| n01 | VBTC File Skip Forward |
| n02 | VBTC File Skip Backward |
| n03 | VBTC Read File Marks Forward |
| n04 | VBTC Read File Marks Backward |
| n05 | VBTC Block Skip File Marks Forward |
| n06 | VBTC Block Skip File Marks Backward |
| n10 | VBTC Write Forward |
| n11 | VBTC Read Forward |
| n12 | VBTC Read Backward |
| n13 | VBTC Block Skip Forward |
| n14 | VBTC Block Skip Backward |
| n15 | Pipeline write forward |
| n16 | Pipeline error |
| n20[1] | VBTC Erase Gap |
| n25 | VBTC End-of-tape Sensing |
| n30 | VBTC Forced Parity Error |
| n50 | VBTC Write Files with Fixed Length Records |
| n51 | VBTC Read Files with Fixed Length Records |
| n52 | VBTC Write Files with Variable Length Records |
| n53 | VBTC Read Files with Variable Length Records |
| n60 | VBTC Write Chained Mode Data Files |
| n61 | VBTC Read Chained Mode Data File |

[1] *n* varies from 3 to 5; 3 is 800 bpi, 4 is 1600 bpi, and 5 is 6250 bpi.

The following execution times, which are for an STC 2920 drive at 1600 BPI, are approximate. Other drives and densities will affect these figures. Specifying a block size larger than the default changes the runtimes for these subtests. All times except *n*00 include rewind time from the tape position at the end of the last subtest. If subtests *n*50-*n*53 are run to the end of the tape, then subtest times are appropriately lengthened, depending on the length of the tape.

**Table dev5200–7, Execution Times for $n = 4$**

| SUBTEST | TIMES (min/sec) |
|---|---|
| 400[1] | 00:11 |
| 401 | 00:12 |
| 402 | 00:19 |
| 403 | 00:10 |
| 404 | 00:20 |
| 405 | 00:10 |
| 406 | 00:19 |
| 410 | 00:30 |
| 411 | 00:04 |
| 412 | 01:44 |
| 413 | 00:28 |
| 414 | 01:00 |
| 415 | 00:25 |
| 416 | 00:16 |
| 420 | 00:49 |
| 425 | time to end of tape |
| 430 | 00:46 |
| 450 | 00:22 |
| 451 | 00:35 |
| 452 | 02:00 |
| 453 | 03:15 |
| 460 | 02:18 |
| 461 | 02:21 |

[1] Execution times for the 300 and 500 series subtests vary from those shown for the 400 series.

### Subtest n00, VBTC Write Tape Mark

Subtest *n*00 rewinds the tape and then writes a pattern of 100 tape marks. Media errors are ignored unless the count exceeds 5.

> **NOTE**
>
> In this subtest and all the following subtests, $n$ varies from 3 to 5 based on tape density; 3 is 800 bpi, 4 is 1600 bpi, and 5 is 6250 bpi.

### Subtest n01, VBTC File Skip Forward

Subtest *n*01 rewinds the tape and then uses the file *skip forward* function to skip 90 tape marks written by Subtest *n*00. The subtest ignores media errors unless the count exceeds 10.

### Subtest n02, VBTC File Skip Backward

Subtest *n*02 rewinds the tape, skips 90 tape marks written by Subtest *n*00 in the forward direction, and then skips 80 tape marks in the reverse direction. This subtest ignores media errors unless the count exceeds 10 for tape motion in either direction.

### Subtest n03, VBTC Read File Marks Forward

Subtest *n*03 rewinds the tape, and then reads 90 tape marks written by Subtest *n*00 with a *block read* function in the forward direction. Media errors are ignored unless the count exceeds 10.

### Subtest n04, VBTC Read File Marks Backward

Subtest *n*04 rewinds the tape, uses the *file skip forward* function to skip 90 tape marks written by Subtest *n*00, and reads 80 tape marks backward, using the *block read backward* function. This subtest ignores media errors unless the count exceeds 10 in either direction.

### Subtest n05, VBTC Block Skip File Marks Forward

Subtest *n*05 rewinds the tape and then uses the *block skip forward* function to skip over 90 tape marks written by Subtest *n*00. The subtest ignores media errors unless the count exceeds 10.

### Subtest n05, VBTC Block Skip File Marks Backward

Subtest *n*06 rewinds the tape, uses the *file skip forward* function to skip 90 tape marks written by Subtest *n*00, and then it skips 80 tape marks backward using the *block skip backward* function. The subtest ignores media errors unless the count exceeds 10 in either direction.

### Subtest n10, VBTC Write Forward

Subtest *n*10 rewinds the tape and then writes a data pattern composed of 200 records, each containing incrementing byte values, modulo 256, beginning with zero, followed by 5 tape marks. The number of bytes entered at the test parameter prompt `Fixed Length Block Size` determines the record size; the default record size is 512 bytes. Media errors and block size errors are ignored unless the total count exceeds 10. Errors writing the tape marks are ignored if at least one is valid.

### Subtest n11, VBTC Read Forward

Subtest *n*11 first rewinds the tape and then reads forward 190 records written by Subtest *n*10. This subtest ignores media, block size, and data errors unless the total error count exceeds 10.

### Subtest n12, VBTC Read Backward

Subtest *n*12 rewinds the tape, skips the pattern written by Subtest *n*10 using the *file skip* function, and then the pattern is read backward for 190 records. The subtest ignores media, block

size, and data errors unless the total error count exceeds 10.

### Subtest n13, VBTC Block Skip Forward

Subtest *n*13 rewinds the tape and then skips over 190 tape blocks written by Subtest *n*10 using the *block skip* function. Media errors are ignored unless the error count exceeds 10.

### Subtest n14, VBTC Block Skip Backward

Subtest *n*14 rewinds the tape, skips the pattern written by Subtest *n*10 using the *file skip forward* function, and then it skips 190 tape blocks using the *block skip backward* function. Media errors are ignored unless the error count exceeds 10.

### Subtest *n*15, Piped Write Forward

Subtest *n*15 rewinds the tape and performs 200 pipelined writes. Correctable errors are ignored but will result in momentary interruption of the command pipe. Uncorrectable errors are retried upto a maximum of 10 times. The record size is taken from the block size option in the user options.

### Subtest n16, Piped Command Error

Subtest *n*16 rewinds the tape and writes one 65,536 byte record. The tape is then rewound. Next, two 16,384 byte reads are pipelined. The test verifies that the first read ends with a read underrun condition and the second read was not executed.

### Subtest n20, VBTC Erase Gap

Subtest *n*20 rewinds the tape, performs at least 50 erase-gap functions, and then performs a *block skip backward* function. The *block skip backward* function is timed to assure that the long gap exists. The subtest ignores media errors when erasing unless the count exceeds 5.

### Subtest n25, VBTC End-of-Tape Sensing

Subtest *n*25 rewinds the tape and performs a *write forward* operation until the EOT marker is read. This subtest executes only when **y** to the `Run EOT Subtest n25` test parameter prompt.

### Subtest n30, VBTC Force Parity Error

Subtest *n*30 rewinds the tape and writes a data pattern with the controller set to force parity errors. The forced errors are verified. The subtest ignores media and block size errors unless the total count exceeds 10.

Then the subtest rewinds the tape and a valid data pattern is written without forced errors. Media and block size errors are ignored unless the total count exceeds 10. Then the subtest rewinds the tape again and reads with forced parity errors. The subtest ignores media, block size,

and data errors unless the total count exceeds 10.

### Subtest n50, VBTC Write Files with Fixed Length Records

Subtest *n*50 rewinds the tape, and then writes 10 files of data patterns. Each file consists of 200 fixed-length data blocks and is terminated with a tape mark. A second tape mark follows the last file. Each data block, which has its length determined by the number entered for the test parameter prompt `Fixed Length Block Size` (default 512 bytes), consists of increasing byte values, modulo 256, beginning with the block number (modulo 256) in the file. The first byte in each block is an exception; it contains the alternating values 0xaa and 0x55. When the end of tape marker is read the first byte contains the value 0xff.

The subtest recovers media errors by using a backspace plus erase gap plus rewrite technique. Any media error is retried the number of times specified in the test parameter prompt `Media Error Retry Count` (the default is 5).

The end of tape causes orderly (but premature) termination of the written data structure compatible with Subtest *n*51. Entering **y** to the test parameter prompt `Run Subtests n50-n53 to EOT` causes the subtest to write as many files of data as are required to fill the tape.

The error statistics which are displayed at the end of the subtest, can also be displayed as the errors occur by entering **y** to the `Display Each Media Error` test parameter prompt.

### Subtest n51, VBTC Read Files with Fixed Length Records

Subtest *n*51 rewinds the tape and then reads and verifies the data structure written by Subtest *n*50. The subtest recovers media errors by using a backspace plus reread technique. Any media error is retried the number of times specified for the test parameter prompt `Media Error Retry Count` (default is 5). The subtest detects the end of tape (if any) by reading software marks written by Subtest*n*50.

The error statistics which are displayed at the end of the subtest, can also be displayed as the errors occur by entering **y** to the `Display Each Media Error` test parameter prompt.

### Subtest n52, VBTC Write Files with Variable Length Records

Subtest *n*52 rewinds the tape and then writes one file to the tape. The file is terminated with two tape marks. The data in the file consists of records of various lengths, from 18 to 128*1024-1 bytes. Each record contains increasing byte values, modulo 256, beginning with the block size. The first byte in each block is the end of tape marker, which normally has a value of 0xaa. The first byte has a value of zero after the end of tape marker is read.

The subtest recovers media errors by using a backspace plus erase gap plus rewrite technique. Any media error is retried the number of times specified in the test parameter prompt `Media Error Retry Count` (default is 5).

The end of tape causes orderly (but premature) termination of the written data structure compatible with the read Subtest *n*53. If **y** is entered to the test parameter prompt `Run Subtests n50-n53 to EOT`, the subtest writes as many files of data as are required to fill the tape.

The error statistics which are displayed at the end of the subtest, can also be displayed as the errors occur by entering `y` to the `Display Each Media Error` test parameter prompt.

### Subtest n53, VBTC Read Files with Variable Length Records

Subtest *n*53 rewinds the tape and then reads and verifies the data structure written by Subtest *n*52. The subtest recovers media errors by using a backspace plus reread technique. The subtest detects the end of tape by reading software marks (if any) written by Subtest *n*52.

The error statistics which are displayed at the end of the subtest, can also be displayed as the errors occur by entering `y` to the `Display Each Media Error` test parameter prompt.

### Subtest n60, VBTC Write Chained Mode Data Files

Subtest *n*60 rewinds the tape and then writes one file to the tape. The file consists of several records of various sizes written in chained mode. Two file marks follow the data. The subtest recovers media errors by using a backspace plus erase gap plus rewrite technique. Any media error are retried the number of times specified in the test parameter prompt `Media Error Retry Count` (default is 5).

The error statistics which are displayed at the end of the subtest, can also be displayed by entering `y` to the `Display Each Media Error` test parameter prompt.

### Subtest n61, VBTC Read Chained Mode Data File

Subtest *n*61 rewinds the tape and then reads and verifies the data structure written by Subtest *n*60. The subtest recovers media errors by using a backspace plus reread technique. Any media error are retried the number of times specified in the test parameter prompt `Media Error Retry Count` (default is 5).

The error statistics which are displayed at the end of the subtest, can also be displayed by entering `y` to the `Display Each Media Error` test parameter prompt.

## Class 6 Subtest

There is only one Class 6 subtest.

### Subtest 600, STC 2922 Speed Select

Subtest 600 rewinds the tape and selcects 100ips (select code 0x08). It then performs 256 pipelined writes of 65,536 byte records and measures the average tape velocity acheived. Correctable errors are ignored but will result in momentary interruption of the command pipe. The above sequence is repeated for 50 ips (select code 0x01). The ratio from 100 to 50 ips must be 2 to 1 or greater.

# Error Messages

**Erase Gap Failed**
The gap erase function did not succeed in erasing a gap on the tape.

**Exception for VIOP d from recv_viop**
{error message detail}
An error has occurred using the SPU/VIOP interface to wait for a signal from the VIOP that the command is finished. Refer to "Exception from setup_viop" for message detail.

**Exception from load_viop**
{error message detail}
An error has occurred while bootstrapping the VIOP. Make sure that the VIOP program image file *dev5200.x00* is available and readable. It must reside either in the current directory or */mnt/test*. Refer to "Exception from setup_viop" for message detail.

**Exception from send_viop (*n*)**
{error message detail}
An error has occurred using the SPU/VIOP interface to signal the VIOP that a command is ready. Refer to "Exception from setup_viop" for message detail.

**Exception from setup_viop**
{error message detail}
An error occurred when preparing to access the VIOP. When this error occurs, the message detail may be:

**Table dev5200–8, *setup_viop* Exception Error Messages**

| TEXT | MEANING |
|---|---|
| HARD ERROR | Hardware error |
| VIOP BUS ERROR | Controller address error |
| VIOP CACHE ERROR | VIOP hardware error |
| VIOP PBUS ERROR | VIOP hardware error |
| MMIO ERROR | Main memory error |
| VMEbus ERROR | Hardware error |
| TIMEOUT | Possible hardware error |

**Exception from start_viop**
{error message detail}
An error has occurred while starting the VIOP. Refer to "Exception from setup_viop" 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 or tape drive status dump, if applicable}
There is a problem in VIOP processing. The following messages may appear:

## Table dev5200–9, VIOP Processing Error Messages

| TEXT | MEANING |
|---|---|
| TIME OUT | no response to device command |
| CHAINING OVERRUN | program and tape drive got out of sync chaining |
| MEDIA ERROR | tape error |

**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.

**VBTC exec level command:** *"cccccccccc" (0xnn)*
This message gives a description of the command, where *cccccccccc* is the verbal description of the command.

**VBTC exec level command: expected** *"cccccccccc" (0xnn)* **actual** *"cccccccccc" (0xnn)*
This message gives the expected and actual command, where *cccccccccc* is the verbal description of the command.

**VBTC exec level unit:** *nn*
This message gives the unit number in the execution level of the controller.

**VBTC exec level unit: expected** *nn*, **actual** *nn*
This message indicates a discrepancy in the unit number in the execution level of the controller.

**VBTC cstat[1]** *0xnn* **means:**
{error message detail}
This indicates a controller error status byte at offset 0x18 of controller. (The error message detail appears in the next entry.)

**VBTC cstat[1]: expected** *0xnn*, **actual** *0xnn*, **differences:**
{error message detail}
This indicates a discrepancy in the controller status at offset 0x18. The interpretation may be:

```
LAST BYTE [IN]VALID
[NO] VMEbus DMA TIMEOUT
[NO] READ PARITY ERROR
[NO] TAPE MARK
PENDING COMMAND [NOT] ABORTED
TAPE ERROR SUMMARY BIT [NOT] SET
```

**VBTC cstat[2]** *0xnn* **means:**
{error message detail}
This gives the interpretation of the formatter status byte at controller offset 0x19. (Possible interpretations appear in the next message entry.)

**VBTC cstat[2]: expected *0xnn*, actual *0xnn*, differences:**
{error message detail}

This message indicates a discrepancy in the formatter status byte at offset 0x19. The error message detail may be:

```
FORMATTER COMMAND [IN]COMPLETE
FORMATTER COMMAND REJECTED
FORMATTER COMMAND ACCEPTED
[NO] DATA OVERRUN
[NO] DATA CHECK
[NO] FORMATTER PROM ERROR
[NO] FORMATTER CORRECTABLE ERROR
[NO] FORMATTER DATA BUS ERROR
```

**VBTC exec level density:** *ddddd (nn)*

This messages gives the name and number of the density setting.

**VBTC exec level density: expected** *ddddd (nn)*, **actual** *ddddd (nn)*

This message gives the names and numbers of the expected and actual density.

**VBTC dstat[2]** *0xnn* **means:**
{error message detail}

This gives the interpretation of the drive status byte at offset 0x22. (The interpretations appear in the next entry.)

**VBTC dstat[2]: expected** *0xnn*, **actual** *0xnn*, **differences:**
{error message detail}

This indicates a discrepancy in the drive status byte at offset 0x22. The message detail may be:

```
DIAG MODE LATCH [NOT] SET
[NO] UCE CONDITION
[NO] WRITE TAPE MARK CHECK
[NO] END DATA CHECK
[NOT] MTE CONDITION
[NO] VEL ERROR
```

**VBTC dstat[3]** *0xnn* **means:**
{error message detail}

This gives the interpretation of the drive status byte at offset 0x23. The interpretation may be:

```
[NO] CRC ERROR
```

**VBTC dstat[3]: expected** *0xnn*, **actual** *0xnn*, **differences:**
{error message detail}

This indicates a discrepancy in the drive status byte at offset 0x23. The interpretation may be:

```
[NO] CRC ERROR
```

**VBTC dstat[6] *0xnn* means:**
{error message detail}

This gives the interpretation of the drive status byte at offset 0x26. (The interpretations appear in the next entry.)

**VBTC dstat[6]: expected *0xnn*, actual *0xnn*, differences:**
{error message detail}

This indicates a discrepancy in the drive status byte at offset 0x26. The interpretation may be:

```
DRIVE [NOT] READY
DRIVE [NOT] ONLINE
FILE [NOT] PROTECTED
[NOT] AT BEGINNING OF TAPE
[NOT] BACKWARD MOTION
[NOT] HIGH DENSITY
[NOT] PAST END OF TAPE
```

**VBTC dstat[7] *0xnn* means:**
{error message detail}

This gives the interpretation of the drive status byte at offset 0x27. The message may be:

```
[NOT] WRITE FUNCTION
```

**VBTC dstat[7]: expected *0xnn*, actual *0xnn*, differences:**
{error message detail}

This indicates a discrepancy in the drive status byte at offset 0x27. The message detail may be:

```
[NOT] WRITE FUNCTION
```

**VBTC status *0xnn* means:**
{error message detail}

This gives the interpretation of the controller status byte at offset 0x10. The message detail appears in the next entry.

**VBTC status: expected *0xnn*, actual *0xnn*, differences:**
{error message detail}

This indicates a discrepancy in the controller status byte at offset 0x10. The message detail may be:

```
BLOCK MODE
BUFFER A ACTIVE
BUFFER B ACTIVE
CHAIN MODE
COMMAND [NOT] EXECUTING
COMMAND [NOT] PENDING
DIAGNOSTICS [NOT] ENABLED
FORCE PARITY ERROR
INTERRUPT [NOT] ENABLED
[NO] INTERRUPT REQUEST
USE NORMAL PARITY
```

**Pattern error at offset** *0xnnnnnn* **in buffer: expected** *0xnn* **actual** *oxnn [0xnn 0xnn ...]*
This message indicates an error in data has been detected. The context following the erroneous byte may also be printed.

**Retry count for this record:** *nn*
This message gives the number of retries required to overcome a media error. This message is generally appended to other messages.

**Tape block size error: Expected size:** *nn*, **actual size:** *nn*
This message indicates an erroneous tape block size.

THIS PAGE INTENTIONALLY LEFT BLANK

# dev5500

# VMEbus Ethernet Controller Test

## Overview

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

## Prerequisites and Required Equipment

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

**Table dev5500–1, Hardware Requirements**

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

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

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

## Test Invocation

The *dev5500* test executes under the Diagnostic Shell (*dshell*) and supports all the features of the *dshell*. The *dshell* permits tests to be initiated in any order.

To invoke the *dev5500* test, use the procedure shown in the following figure. All responses in **boldface** are entered by the user. The prompts and responses appear sequentially on the screen, one line at a time. All prompts and responses are shown in one figure for convenience.

## Figure dev5500–1, Test Invocation Sequence

```
(spu)>  cd /mnt/test RETURN
(spu)>  sysreset RETURN
(spu)>  mminit -s RETURN
(spu)>  dshell RETURN

CONVEX DIAGNOSTICS SHELL

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

**NOTE**

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

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

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

**CAUTION**

The user response, **initall**, is typically required if the *initall* utility has not been run since the last powerup. 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 if 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 dev5500–2, Alternate Test Invocation Sequence**

```
(spu)> cd /mnt/test RETURN
(spu)> initall RETURN
(spu)> dshell RETURN
CONVEX DIAGNOSTICS SHELL
: test dev5500 [-c [class number(s)]] [-s [subtest number(s)]] [+> filename] RETURN
```

## Test Parameter Menu

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

The following figure shows all prompts, with their possible answers (in brackets [ ]), and their default answers (in parenthesis ( )). The prompts and responses in the following figure appear sequentially on the screen, one line at a time. All possible prompts and responses are shown in the following figure for convenience.

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

**Table dev5500–2, Getting Help During Test Parameter Entry**

| CHARACTER | DESCRIPTION |
|---|---|
| ? | Displays this help menu |
| h | Provides help for a specific prompt |
| i | Displays the lioconfig file |

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

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

## Figure dev5500–3, *dev5500* Test Parameter Menu

```
                    ENTER TEST PARAMETERS

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


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


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

Device 0 = user defined configuration
```

At any time during the test parameter sequence, several options are available as denoted at the top of the Test Parmeter 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

**?** — Request help for the current prompt (if available)

**^** — Return to the previous prompt

## Prompt Explanations

A description of the meaning of each prompt follows:

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

**Figure dev5500–4, Sample Test Parameter Summary**

```
                TEST PARAMETER SUMMARY

Device selection                                  : 0
VIOP                                              : 5
VMEbus Chassis                                    : 0
Controller Offset in VMEbus                       : 0x8000
Controller Interrupt                              : 5
Subtest loop count                                : 100
Transceiver connected to active ethernet?         : y
Run machine <-> machine comm test?                : n
Enter OK, or :NN to return to question NN         : OK
```

# Hardware Initialization Sequence

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

- A sysreset is performed
- Main memory is allocated for the test
- SPU windows to main memory are initialized
- SPU local test variables are initialized
- The VIOP is booted and loaded
- A driver on the VIOP is started
- VIOP local test variables are initialized

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

# Class Descriptions

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

**Table dev5500–3, *dev5500* Test Classes**

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

## Class 1 Subtest

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

**Table dev5500–4, Class 1 Subtest**

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

### Subtest 100, Controller to Host Access

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

## Class 2 Subtests

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

**NOTE**

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

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

**Table dev5500–5, Class 2 Subtests**

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

### Subtest 200, Real Physical Slot Xmit/Recv

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

### Subtest 201, Modified Physical Slot Xmit/Recv

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

### Subtest 202, Broadcast Slot Xmit/Recv

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

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

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

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

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

### Subtest 205, Foreign Address Rejection Xmit/Recv

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

### Subtest 206, Scatter/Gather Operation

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

## Class 3 Subtest

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

**Table dev5500–6, Class 3 Subtest**

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

### Subtest 300, Real Physical Slot Only Xmit/Recv

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

## Class 4 Subtest

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

**NOTE**

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

**Table dev5500–7, Class 4 Subtest**

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

**Subtest 400, Machine to Machine Xmit/Recv**

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

# Subtest Completion Messages

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

**Figure dev5500–5, Sample Subtest Completion Messages**

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

# Subtest Error Messages

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

## Figure dev5500–6, Sample Error Message

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

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

The following error messages are for all subtests:

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

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

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

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

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

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

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

A valid message could be the following:

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

or

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

## Test End Message

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

**Figure dev5500–7, Sample End Message**

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

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

# Appendix A
# Reporting Problems

## A.1 Overview

The *contact* utility is the recommended way to report minor hardware deficiencies and technical documentation problems to the Technical Assistance Center (TAC). This utility is an interactive tool that prompts the user for the information to properly file a problem report.

> **NOTE**
>
> The *contact* utility is not intended for requesting customer service for hardware failures. To restore your CONVEX equipment to operational status, faster service can be obtained by directly telephoning the TAC (refer to "Technical Assistance" in the preface).

To use the *contact* utility, there must be a phone connection to the TAC. A UNIX-to-UNIX Communication Protocols (UUCP) allows communication between UNIX systems by either dial-in or hard-wired communication lines. For more information, refer to *uucp(1)* or to the *info(1)* entry in the UNIX man pages.

The name and version number of the product involved is required. Use the *vers* command to ascertain the program or utility name and version. The syntax for the command is **vers** *filename*, where *filename* is the full pathname of the program. If the full pathname of the program is not known, enter **which** *program*. For more information, refer to the *vers(1)* and *which(1)* entries in the UNIX man pages.

## A.2 Information Required to Report a Problem

The *contact* utility requires the following information:

1. The customer name, title, phone number, and corporate name
2. The hardware nomenclature, part number, and revision level, or the technical manual name, document number, and version

> **NOTE**
>
> Use *vers* and *which* to identify product name and version.

3. A short (one line) summary of the problem
4. The more information provided, the more quickly the problem can be isolated and solved. At a minimum, include a detailed description of the problem (including page references, if applicable), the source code, and a stack backtrace whenever possible.

> **NOTE**
>
> See the *adb(1)* or *csd(1)* man pages for information on obtaining stack backtraces.

5. The priority of the problem, selected from a list of six levels
6. Instructions on how to reproduce the problem, including the command syntax used, any flags invoked, or anything else attempted to make the program run
7. Any other comments about the problem or files to be submitted

The *contact* user has a chance to review and edit the report prior to submitting it. If the user decides to delay submitting the report, the session can be aborted. The report is automatically saved in the user's top-level directory in a file named *dead.report*.

See the following figure for a sample *contact* session. User input is in bold lettering, and the system response is in monospace type.

## Figure A–1, Sample *contact* Session

```
%contact RETURN
Welcome to contact version 0.11 ()

Enter your name, title, phone number, and corporate name (^D to terminate)
> Margaret Atwood, systems programmer, 814-4444, University r
> of Chicago RETURN
> CTRL-D

Enter the name of the product involved
> CONVEX UNIX Programmer's Manual, Part I RETURN

Enter the version number (in the form X.X or X.X.X.X) of the product
> Revision 4.0 RETURN

Enter a short (1 line) summary of the problem
> The finger command manual page lists nonexistent bug RETURN

Enter a detailed description of the problem (^D to terminate)
> The finger(1) man page says, under the BUGS section, that "Only the first
line of the .project file is printed."  Happily, this is not true! RETURN
> CTRL-D

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.
> 4 RETURN

Enter the instructions by which the problem may be reproduced (^D to terminate)
> a) put more than one line in .project RETURN
> b) read the man page for finger(1) RETURN
> CTRL-D

Enter any comments that are applicable (^D to terminate) RETURN
> CTRL-D

Do you have any suggestions or comments on the documentation that you
referenced when you were trying to resolve your problem (for example,
additions, corrections organization, accessibility)? (^D to terminate)
> The man page should be updated. RETURN
> CTRL-D

Are there any files that should be included in this report (yes | no)?
> no RETURN

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.
> 3 RETURN

Problem report submitted.
%
```

THIS PAGE INTENTIONALLY LEFT BLANK

# Index

## Numeric



## A



## B



## C



## D

## E



## F



## H



## I

## K



## L



## M



## N



## O



## P



## R

## S



## T

## U



## V



## W



## X

**CONVEX PBUS I/O System**
**Diagnostics Manual**
Document No. 760-000750-203, Second Edition

## Reader's Forum

You are invited to submit comments concerning the clarity and service of this manual. Constructive critical comments are most welcome, and will help us continue in our efforts to generate quality customer documentation. Please list the document page number with your questions and comments.

_____

_____

_____

_____

_____

_____

_____

_____

**From:**

Name_____ Title_____

Company_____ Date_____

Address and Phone No._____

**FOR ADDITIONAL INFORMATION OR DOCUMENTATION:**

| Location | Phone Number |
|---|---|
| In Texas | (214)952-0200 |
| Other continental locations | 1(800)952-0379 |
| Outside continental US | Contact local 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

CUSTOMER SERVICE
CONVEX Computer Corp.
P.O. Box 833851
Richardson, TX 75083-3851

(Fold Here Second)

(Tape or Staple)