CONVEX VMEbus Async Controller (*dev5300*) Diagnostics Manual

Document No. 760-003230-000

First Edition
May 1991

CONVEX Computer Corporation
Richardson, Texas USA

CONVEX VMEbus Async Controller (dev5300) Diagnostics Manual
Order No. DHW-244
First Edition

Printed in the United States of America

Revision Sheet

CONVEX VMEbus Async Controller (dev5300) Diagnostics Manual

Edition	Document No.	Date	Description
First	760-003230-000	May 1991	First release. Contains the *dev5300* diagnostic test information from the *CONVEX PBUS I/O Systems Diagnostics Manual.*

THIS PAGE INTENTIONALLY LEFT BLANK

Table of Contents

Appendixes

List of Tables

List of Figures

THIS PAGE INTENTIONALLY LEFT BLANK

Preface

Purpose and Intended Audience

This manual explains how to run the *dev5300* diagnostic, which checks the CONVEX VMEbus Async (68020-based) TTY/Printer Controller and its ability to operate in the CONVEX VMEbus I/O environment. This document is not a tutorial, but rather a reference for the users of the *dev5300* diagnostics, including field service and manufacturing test personnel, as well as the diagnostics sustaining staff. In addition, CONVEX customers can use this manual to execute the *dev5300* diagnostic.

Scope

This manual applies to all CONVEX computers.

Organization

This document consists of the following:

- **Chapter 1. Diagnostics Environment**—Introduces the theories and concepts that underlie I/O diagnostics on CONVEX machines as well as the basic overview, philosophy, and structure of I/O diagnostics.
- **Chapter 2. EGOS Overview**—Provides a brief overview of the Event Governed Operating System (EGOS) and how it relates to device and peripheral diagnostics testing.
- **Chapter 3. Dshell Overview**—Provides a brief overview of and a general introduction to the *dshell* utility.
- **Chapter 4. VMEbus Async Controller Test (*dev5300*)**—Describes how to operate the diagnostic, including prerequisites, test invocation, hardware initialization sequence, and class descriptions. It also describes the VMEbus Async EPROM self-tests, the debug monitor, interactive debugger commands, and it gives examples of error messages.
- **Appendix A. Reporting Problems**—Provides an example of the CONVEX *contact* utility for reporting minor software and hardware problems.

Notational Conventions

The notational conventions used in this text are listed below:

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

Warnings

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

WARNING

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

CAUTION

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

NOTE

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

Associated Documents

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

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

Ordering Documentation

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

The product number for this manual is DHW-244.
The document number for this manual is 760-003230-000.

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

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

Technical Assistance

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

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

Reader's Forum

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

Chapter 1
Diagnostics Environment

1.1 Overview

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

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

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

1.2 Test Program Naming Conventions

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

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

1.2.1 Test Program Categories

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

Table 1–1, Test Program Categories

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

1.2.2 Test Program Types

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

Table 1–2, Test Program Types

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

1.2.3 Test Program Device Types

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

Table 1–3, Test Program Device Types

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

1.2.4 Examples of Test Program Names

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

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

Table 1–4, Example Test Program Names

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

THIS PAGE INTENTIONALLY LEFT BLANK

Chapter 2
EGOS Overview

2.1 Overview

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

2.2 Purpose of EGOS for Diagnostic Testing

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

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

2.3 EGOS for the Multibus Interface

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

2.4 EGOS for HSP Interface, HSP EGOS

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

2.5 EGOS for VME Interface, VIOP EGOS

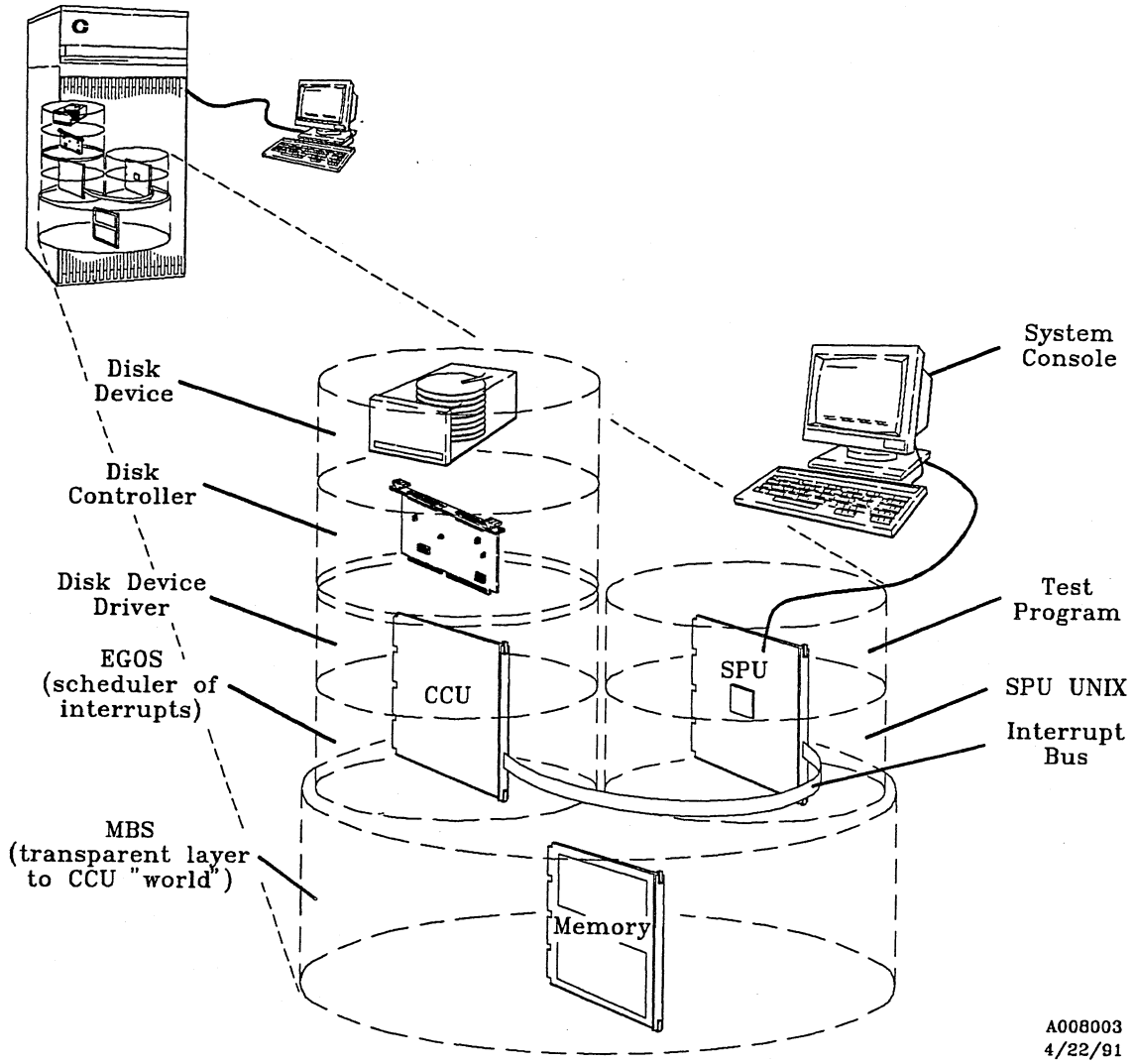
The VME interface version of EGOS is designed with a scheduler for the VIOP and is called VIOP EGOS. VIOP EGOS supports event driven device drivers as well as process type device drivers. VIOP EGOS utilizes a *sleep/wakeup* type of process control that improves efficiency of the device driver and makes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

2.6 EGOS Position in the Environment

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

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

Figure 2–1, EGOS' Position in the Environment



THIS PAGE INTENTIONALLY LEFT BLANK

Chapter 3
Dshell Overview

3.1 Overview

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

3.2 Diagnostic Shell (*dshell*) Overview

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

The *dshell* has two basic functions:

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

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

Table 3–1, *dshell* Commands

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

3.3 Syntax Help for *dshell* Commands

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

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

```
: loop
Proper syntax is:

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

THIS PAGE INTENTIONALLY LEFT BLANK

Chapter 4
VMEbus Async Controller Test (*dev5300*)

4.1 Overview

This diagnostic provides a functional test for the CONVEX VMEbus Async (68020 based) TTY/Printer Controller (part number 410-001193-200) and its ability to operate in the CONVEX VMEbus I/O environment. The diagnostic verifies the supported controller commands. In addition to the diagnostic subtests within *dev5300*, EPROM self-tests are performed every time the controller is reset or powered on. The EPROM self-tests are described herein.

A VMEbus Async Controller and panel are required; loopback cables (Systech loopback cable – CONVEX part number 603-030012-200) are required if the external loopback serial device tests are to be executed. The diagnostic also provides a debug monitor similar to *vioputil*, which can be enabled on errors or may be invoked from the Service Processor Unit (SPU) prompt. Refer to the "Debug Monitor" section within this test description for more information.

The VMEbus Input Output Processor (VIOP) uses the Event Governed Operating System (EGOS). Service Processor/VIOP communication is accomplished via the Message Based System (MBS) as used by ConvexOS. The Service Processor Unit (SPU), SPU peripherals, main memory subsystem, VIOP, and VMEbus Control Unit (VBCU) are all assumed to be fault free prior to the execution of this diagnostic. Although there are diagnostic messages generated by faults in these subsystems, it is not the intent of this diagnostic to test these subsystems.

4.2 Related Documents

This test description is intended as a reference for users who are familiar with the VIOP, VBCU, VMEbus, and the CONVEX I/O system. The user should also be familiar with RS-232 serial devices, Direct Memory Access (DMA) operations and the 68020 microprocessor. Additional information on the VMEbus Async Controller can be found in the following CONVEX documents (part numbers are listed if available):

- IOP Hardware Specification
- VME IOP Difference Document
- VBCU Hardware Specification
- VME Async/Printer Controller Functional Hardware Specification (CONVEX part number 410-001193-000)
- VME Gate Array Functional Specification (CONVEX part number 182-000135-000)

4.3 Prerequisites and Required Equipment

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

Table 4–1, Hardware Requirements

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

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

4.4 Test Invocation

The *dev5300* 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 *dev5300* test, use the procedure shown in the following figure.

Figure 4–1, Initial Test Invocation Sequence

```
(spu)> cd /mnt/test RETURN
(spu)> sysreset RETURN
(spu)> mminit -s RETURN
(spu)> dshell RETURN
:test dev5300 [-c [class number(s)]] [-s [subtest number(s)]] RETURN
```

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.

NOTE

Use the previous test invocation sequence for the initial invocation of *dev5300* or for when the state of the machine is unknown. Also, the previous invocation sequence should be used if any hard errors have occurred since the last system initialization.

NOTE

After entering **dshell**, specific *dshell* parameters may be changed. Refer to the "Dshell Overview" chapter of this manual for more information.

The format for the *test* command is as follows:

test dev5300 [*option* ...]

where *option* is one of the following:

-c *class-number*	Execute one or more specific classes of subtest(s)
-f *filename*	Use *filename* as the parameter save file. If this option is omitted, the parameter file used is */tmp/dev5300.tmp*.
-d	Go to debug mode only; no tests are executed
-s *subtest-number*	Execute one or more individual subtests
-V	Print the version (build date) of this test

Entering only **test dev5300** executes all *dev5300* subtests sequentially.

NOTE

The following alternate test invocation procedure is optimal when invoking *dev5300* multiple times. Using this invocation sequence insures that the test is invoked and executed with all of the set-up parameters supplied when the test was last executed with the initial invocation sequence.

Figure 4–2, Alternate Test Invocation Sequence

```
(spu)> cd /mnt/test RETURN
(spu)> sysreset RETURN
(spu)> mminit -s RETURN
(spu)> dshell RETURN
: test dev5300x [-c [class number(s)]] [-s [subtest number(s)]] RETURN
```

The only difference in this alternate invocation sequence is the **x** after **dev5300**. When invoking *dev5300* in this manner, no prompts are displayed. The diagnostic obtains all prompt information from the parameter file created when the initial invocation sequence was performed. Also note that **mminit -s** is only required if the state of the machine is unknown or if hard errors have occurred since the last system initialization.

4.4.1 Test Parameter Menu

Once the test is invoked, a test menu prompt is presented allowing selection of default switches. The following figure shows ***all*** prompts, their possible answers (in brackets [ ]), and their default answers (in parentheses ( )).

Figure 4–3, Test Parameter Menu

```
Test 'dev5300.t'              Fri Nov 10 15:11:04 1989

                      ENTER TEST PARAMETERS

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

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

      PERIPHERAL CONFIGURATION DATA
     CCU    Chassis   Type    CSR   Int
    -------- ------- -------- ------ ---
 1) viop  7     0    ACM-201  0x03c0  7
 2) viop  5     1    ACM-201  0x03c8  6
 3) viop  5     0    ACM-201  0x1100  1

          *** Enter 0 for manual configuration ***

 2: Ioconfig File Unit to Test [1-3,0,?]                (1) -> 0
 3: VIOP Number [3-7,?]                                 (7) -> 7
 4: VME Chassis Number [0-1,?]                          (0) -> 0
 5: CSR of the Controller [0x10-0xffff,?]          (0x03c0) -> 0x03c0
 6: Interrupt Number of the Controller [1-7,?]          (7) -> 7
     Baud Rate Choices:
         8000=  50       4000=  75
         2000=  110      1000=  134.5
         0800=  150      0400=  200
         0200=  300      0100=  600
         0080=  1050     0040=  1200*
         0020=  2000     0010=  2400*
         0008=  4800     0004=  9600*
         0002=  19200*   0001=  38400*
 7: Select Baud Rates Mask (* indicates default) [0x0-0xffff,?]
                                                  (0x57) -> 0x57
```

Figure 4–3, Test Parameter Menu (continued)

```
 8: Port Test Mask (hex), bit 2^n for port n [0x0-0xffff,?]
                                                  (0xffff) -> 0xffff
 9: Enable Debug Monitor [y,n,?]                       (n) -> n
10: Use Defaults for Remaining Parameters [y,n,?]      (y) -> n

  ** Printer On/Off Enable/Disable Options (bit mapped) **
    0x0001: Enable printer ON string before error
    0x0002: Enable printer OFF string after error

11: Select Printer On/Off Mode [0x0-0x3,?]            (0x0) -> 0x3
12: Printer On Character Sequence (before error)
    [<printer on string>,?]                  (\033[?5i  -> \033[?5i
13: Printer OFF Character Sequence (after error)
    [<printer off string>,?]                 (\033[?5i  -> \033[?5i
14: Enable External Loopback Serial Device Tests [y,n,?]  (n) -> y
15: Enable Internal Loopback Serial Device Tests [y,n,?]  (n) -> y
16: Ignore Errors? [y,n,?]                            (n) -> y
17: Select Download File [<filepath>,?]       (dev5300.xx0) -> dev5300.xx0
     Printer Type Choices:
           Data Products - 0
           Centronics    - 1
18: Select Printer Type [0x0-0x1,?]                   (0x1) -> 0x1
19: Select Pattern File for Printer Test [<filepath>,?]
                                             (dev5300.pat) -> dev5300.pat
20: Select Number of Characters per Line [1-132,?]     (80) -> 80
21: Add <LF> after Line [y,n,?]                        (y)  -> y
22: Add <CR> at end of Line [y,n,?]                    (y)  -> y
23: Select number of Lines per Page [1-60,?]           (60) -> 60
24: Select Number of Lines to Print [1-1000,?]         (61) -> 60
25: Enter OK, or :NN to return to question NN [OK]     (OK) -> OK
```

Some questions may be omitted, depending on answers to previous questions. In all cases, questions are numbered sequentially. 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.

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

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

Table 4–2, Getting Help During Test Parameter Entry

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

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

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

Figure 4–4, Sample Test Parameter Summary

```
                        TEST PARAMETER SUMMARY

Select ioconfig file                                          : ioconfig
Ioconfig File Unit to Test                                    : 0
VIOP Number                                                   : 7
VME Chassis Number                                            : 0
CSR of the Controller                                         : 0x03c0
Interrupt Number of the Controller                            : 7
Baud Rates Selected:
         1200    2400     9600    19200
         38400
Select Baud Rates Mask (* indicates default)                  : 0x57
Port Test Mask (hex), bit 2^n for port n                      : 0xffff
Enable Debug Monitor                                          : n
Use Defaults for Remaining Parameters                         : n
Select Printer On/Off Mode                                    :0x0000
         -> Enable printer ON string before error <-
         -> Enable printer OFF string after error <-
Printer On Character Sequence (before error)                  : \033[?5i
Printer Off Character Sequence (after error)                  : \033[?4i
Enable External Loopback Serial Device Tests                                : n
Enable Internal Loopback Serial Device Tests                                : n
Ignore Errors                                                 : n
Select Download File                                          : dev5300.xx0
  Printer Type Selected:  Centronics
Select Printer Type                                           : 0x1
Select Pattern File for Printer Test                          : dev5300.pat
Select Number of Characters per Line                          : 80
Add <LF> after Line                                           : y
Add <CR> at end of Line                                       : y
Select number of Lines per Page                               : 60
Select Number of Lines to Print                               : 61
Enter OK, or :NN to return to question NN                     : OK
```

4.5 Initialization Sequence for *dev5300*

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

- The diagnostic determines if the test was invoked for quick start-up (i.e., **dev5300x**). If so, the diagnostic reads the test parameters from the specified parameter file (default parameter file is */tmp/dev5300.tmp*). If not, the input to the parameters are written to the parameter file (default or user specified).
- The diagnostic checks to see if the Channel Control Unit (CCU) is already loaded with the *dev5300* CCU driver. If the driver is not loaded, the CCU is loaded. After the load completes, the driver is configured for EGOS and then the EGOS probe message starts the driver.

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

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

4.5.1 Prompt Explanations

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

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

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

```
      PERIPHERAL CONFIGURATION DATA
     CCU    Chassis   Type     CSR    Int
   -------- ------- -------- ------ ---
1) viop  7     0    ACM-201  0x03c0  7
2) viop  5     1    ACM-201  0x03c8  6
3) viop  5     0    ACM-201  0x1100  1

       *** Enter 0 for manual configuration ***

2: Ioconfig File Unit to Test [1-3,0,?]                    (1) ->
```

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

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

Enter the CCU slot number for the VMEbus IOP that contains the VMEbus subsystem containing the VMEbus Async Controller.

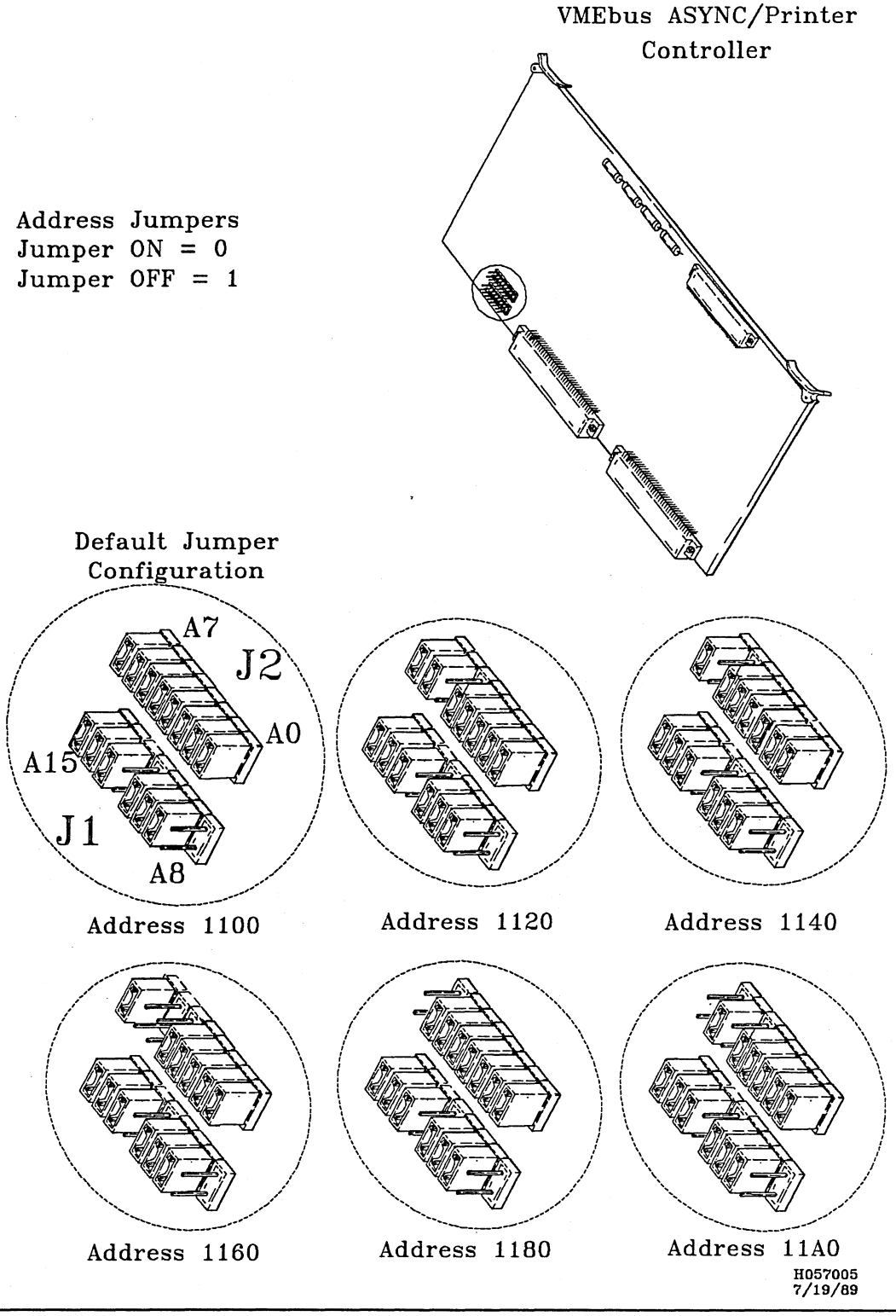
```
4: VME Chassis Number [0-1,?]                              (0) ->
```

Enter the VMEbus chassis number for the chassis where the VMEbus Async Controller is installed. If there is only 1 chassis, the number is most likely 0.

```
5: CSR of the Controller [0x10-0xffff,?]                (0x03c0) ->
```

Enter the low-order four hexadecimal digits of the VMEbus Async Controller board's base address within the VMEbus short address space. This address is controlled by jumpers on the controller. The default address is 0x03c0. The following figure shows the VMEbus Async Controller, the jumpers that control the short address space, and default jumper configuration:

Figure 4–5, VMEbus Async Controller Jumpers



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

Enter the CONVEX VMEbus interrupt number assigned to the VMEbus Async Controller board under test. This value is between 1 and 7, inclusive. The typical interrupt level is 7.

```
Baud Rate Choices:
    8000=  50       4000=  75
    2000=  110      1000=  134.5
    0800=  150      0400=  200
    0200=  300      0100=  600
    0080=  1050     0040=  1200*
    0020=  2000     0010=  2400*
    0008=  4800     0004=  9600*
    0002=  19200*   0001=  38400*
7: Select Baud Rates Mask (* indicates default) [0x0-0xffff,?]
                                                    (0x57) ->
```

This prompt allows specification of the baud rate at which the serial devices are tested. Default baud rates are marked by a asterisk (*).

> **NOTE**
>
> Selection of the options is achieved by entering the hexadecimal mask obtained by ORing the desired options (hexadecimal numbers to the left of the baud rate) together. For example, if the first two baud rates are desired (50 and 75), OR their hexadecimal masks (4000 and 8000) together and enter C000.

```
8: Port Test Mask (hex), bit 2^n for port n [0x0-0xffff,?]
                                                    (0xffff) ->
```

This prompt allows selection of which ports to test.

> **NOTE**
>
> The external loopback tests require that ports be tested in adjacent pairs, i.e., in order to test port 1, port 0 must be enabled as well. The internal loopback tests do not require any such restriction, a single port may be tested.

```
9: Enable Debug Monitor [y,n,?]                        (n) ->
```

This option allows automatic entry to the interactive debugger any time an error is detected within a subtest. The debugger may also be invoked by specifying the **-d** option at test invocation time (i.e., **dev5300[x] -d**).

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

Answering yes in response to this prompt results in the selection of the default value for all remaining prompts. This avoids having to explicitly enter a RETURN in response to all remaining prompts.

```
** Printer On/Off Enable/Disable Options (bit mapped) **
  0x0001: Enable printer ON string before error
  0x0002: Enable printer OFF string after error

11: Select Printer On/Off Mode [0x0-0x3,?]                (0x0) ->
```

This option provides the ability to send a specified character string to the display before or after an error and its data have been displayed. Although any string up to 64 characters may be sent, the intended use is to allow the ability to selectively turn a printer on before an error is displayed and turn a printer off after an error is displayed. This is a paper saving feature when running the test over and over for several hours. Errors are only printed when both the on and off strings are enabled. This assumes a printer is connected to the auxiliary port on the display terminal where the test is running and that the auxiliary port can be turned on and off via an escape character sequence.

```
12: Printer On Character Sequence (before error)
    [<printer on string>,?]                  ( 33[?5i) ->
```

This is the character string used to turn on the printer. A different string can be entered if needed.

```
13: Printer Off Character Sequence (after error)
    [<printer off string>,?]              ( 33[?4i) ->
```

This is the character string used to turn off the printer. A different string can be entered if needed.

```
14: Enable External Loopback Serial Device Tests [y,n,?]       (n) ->
```

This allows external loopback tests for the Universal Asynchronous Receiver Transmitter (UART) and Dual Universal Serial Communications Control (DUSCC) ports to be enabled. If these subtests are enabled, be sure that loopback cables (Systech loopback cable 603-030012-200) are installed in all ports specified for testing.

```
15: Enable Internal Loopback Serial Device Tests [y,n,?]       (n) ->
```

This allows internal loopback tests for the UART and DUSCC to be enabled. Loopback cables are not required.

```
16: Ignore Errors [y,n,?]                                      (n) ->
```

This option causes the diagnostic to log errors but to continue running. Upon completion of a test the error log is printed out (if any errors occurred) but the test will be treated as though it passed. However, a cache controller error, cable error, or bus error on the VIOP will not be put in the error log and will cause the test to fail.

```
17: Select Download File [<filepath>,?]          (dev5300.xx0) ->
```

The download file is the file that is downloaded to the controller for class 2 and class 3 tests. Subtest 200 will download any *b.out* file to the controller but it should be noted that the remaining tests depend upon the proper program being executed on the VMEbus Async Controller.

```
        Printer Type Choices:
            Data Products - 0
            Centronics    - 1
    18: Select Printer Type [0x0-0x1,?]                    (0x1) ->
```

The VMEbus Async Controller supports two types of printer interfaces. A **1** selects the Centronics printer interface and a **0** selects the Data Products printer interface.

```
    19: Select Pattern File for Printer Test [<filepath>,?]
                                                  (dev5300.pat) ->
```

This is the file used by the printer test for the test pattern. The file can be up to 4 Kbytes in size and should not contain carriage returns.

```
    20: Select Number of Characters per Line [1-132,?]      (80) ->
```

This prompt specifies the number of characters to print per line.

```
    21: Add <LF> after Line [y,n,?]                          (y) ->
```

This prompt specifies whether a line feed is added after each line.

```
    22: Add <CR> at end of Line [y,n,?]                      (y) ->
```

This prompt specifies whether a carriage return is added at end of line.

```
    23: Select number of Lines per Page [1-60,?]            (60) ->
```

This prompt asks for the number lines to print per page. After the specified number of lines are printed a form feed is issued to the printer.

```
    24: Select Number of Lines to Print [1-1000,?]          (61) ->
```

This prompt specifies the total number of lines to print in the test.

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

If **OK** is entered, the test parameter menu terminates and all inputs are no longer changeable. To return to any previous prompt, enter the prompt number instead of **OK**.

4.6 VMEbus Async EPROM Self-tests

The VMEbus Async TTY/Printer Controller EPROM contains self-diagnostic routines that are executed immediately after any reset to the board or anytime power is applied. The controller EPROM is copied into instruction RAM in addresses 0x0–0x7fff. The self-tests perform initialization upon reset, downloading of Operating System driver modules, execution of self-tests, and a debug monitor (accessed from an available TTY port of the controller).

Upon reset or power up, the EPROM first disables interrupts, turns off the VMEbus DMA and the printer interface, enables bus timeouts, clears all pending interrupts, and then computes an EPROM check sum. If the check sum computation fails, a 0x55 is displayed on the controller LEDs and the processor stops. If the check sum is correct, the EPROM clears instruction RAM and then examines its VMEbus address as read from the jumpers (see Figure dev5300-5, VMEbus

Async Controller Jumpers). If the controller address is 0x11c0 (see Figure dev5300-6, Jumper Setting for Debug Monitor), the controller enters the Debug Monitor, described in the "Debug Monitor" section of this test description. If the address is not 0x11c0, the EPROM starts execution of the self-tests. These self-tests are described in more detail in the following section.

Upon failure of a self-test, the self-test number is placed in the LED Display Register and displayed on the controller LEDs and the processor stops. If the controller is idle, the controller LEDs cycle. The LEDs are numbered zero through seven from left to right, respectively. The least significant bit is LED zero and the most significant bit is LED seven.

If all self-tests pass, the EPROM enables interrupt A and interrupt B, clears the SYS_FAIL bit in the Reset Register, and then waits for an interrupt.

4.6.1 EPROM Self-Test Descriptions

The following sections describe each VMEbus Async TTY/Printer Controller EPROM self-test.

4.6.1.1 Self-Test 1, Instruction Memory Self-Test

Each byte of the instruction RAM (address 0x0–0x3ffff) is accessed and unique data is written. The memory is then read and verified. The pattern '0xaaaaaaaa/0x55555555' is written and verified throughout memory. The pattern is then inverted and the algorithm repeated.

4.6.1.2 Self-Test 2, Memory Address Self-Test

This self-test quickly verifies longword accessibility in dual port RAM (address 0x40000000–0x4003ffff). A one is walked across the address and unique data is written and verified in dual port RAM.

4.6.1.3 Self-Test 3, DPR Memory Self-Test

Each byte of dual port RAM (address 0x40000000–0x4003ffff) is accessed and unique data is written. The memory is then read and verified. Next, the pattern '0xaaaaaaaa/0x55555555' is written and verified throughout memory. The pattern is then inverted and the algorithm repeated.

4.6.1.4 Self-Test 4, Array Register Self-Test

A walking ones/zeros test is performed on all valid bits on all read/write registers on the board. The registers tested are:

- intr #0 priority
- intr #1 priority
- intr #2 priority
- intr #3 priority
- intr #4 priority
- intr #5 priority
- intr #6 priority
- intr #7 priority
- intr #8 priority
- intr #9 priority
- intr #10 priority
- intr #11 priority
- intr #12 priority
- intr #13 priority
- intr #14 priority
- intr #15 priority
- intr #16 priority
- intr #17 priority
- intr #18 priority
- intr #19 priority
- intr #20 priority
- intr #21 priority
- intr #22 priority
- intr #23 priority
- intr #24 priority
- intr #25 priority
- intr #26 priority
- intr #27 priority
- intr #28 priority
- intr #29 priority
- intr #30 priority
- IER msb
- IER lsb
- FIR msb
- FIR lsb
- timer a base count
- timer a contol reg
- timer b base count
- timer b control reg
- vmebus map reg (msg)
- vmebus map reg (lsb)
- timeout reg
- supervisory base addr
- vmebus intr cntl reg
- vme dma addr modifier
- vme dma base addr reg msb
- vme dma base addr reg lsb
- vme dma local bar msb
- vme dma local bar lsb
- vme dma byte count
- vme dma throttle reg
- io dma 0 addr reg msb
- io dma 0 addr reg lsb
- io dma 0 byte count
- io dma 1 addr reg msb
- io dma 1 addr reg msb
- io dma 1 byte count
- io dma 2 addr reg msb
- io dma 1 byte count
- io dma 2 addr reg lsb
- io dma 3 addr reg msb
- io dma 3 addr reg lsb
- io dma 3 byte count
- DSR

4.6.1.5 Self-Test 5, Interrupt Arbitration Self-Test

The interrupt arbitration self-test verifies the ability of the board to schedule multiple simultaneous interrupts of the same priority. The self-test causes two interrupts of each priority to be requested simultaneously using the Force Interrupt Register (FIR). The self-test verifies that these interrupts are scheduled, according to priority, in a round-robin manner. Interrupt levels zero through seven are verified.

4.6.1.6 Self-Test 6, First Interrupt Self-Test

The first interrupt self-test (self-test 6) sets up the interrupt priority registers, the Interrupt Vector Base Register, and the Interrupt Enable Register (IER). The self-test enables all interrupts in the IER then causes each interrupt using the Force Interrupt Register (FIR). Each interrupt is caused twice successively and verified in the Interrupt Status Register (ISR).

4.6.1.7 Self-Test 7, Second Interrupt Self-Test

The second interrupt self-test (self-test 7) enables all possible 32 interrupts by writing the appropriate bit in the FIR. The interrupt vector, which is generated, is written to memory and verified.

4.6.1.8 Self-Test 8, Timer Self-Test

The functionality of the Timer Control Register for the two timer circuits are verified. All modes of the timers and the proper generation of interrupts are also verified for both timers.

4.6.1.9 Self-Test 9, Timer Counter Self-Test

The timer counter Self-Test walks a one across the Timer Base Count Register to verify the ability of the Present Count Register to properly decrement the timer count value.

4.6.1.10 Self-Test 10, Timer Scale Self-Test

The timer pre-scaler (value that the reference clock is divided by) functionality is verified by counting the number of iterations of a timing loop with scale set at divide by one (this is used as a reference count). The count used in the scale of one is multiplied by the inverse of the scale factor (i.e., if the scale is ½ the count is multiplied by 2, if the scale ¼ the count is multiplied by 4). The number of iterations of the timing loop with the adjusted count and scale is compared to the base count obtained with the initial scale.

4.6.1.11 Self-Test 11, Printer FIFO Self-Test

The printer FIFO is a 1-Kbyte FIFO 8 bits wide. The FIFO is filled and then read — verifying the functionality of the FIFO full and FIFO empty status bits, as well as the data integrity. The FIFO empty printer interrupt is also verified.

4.6.1.12 Self-Test 12, DUSCC Self-Test

The Dual Universal Serial Communication Controls (DUSCCs) are looped back internally and characters written to each transmit register. The self-test verifies that the characters are received as expected in the receive buffer and that the appropriate status is generated. This self-test only tests at the 9,600 baud rate, 8 bits per character with even parity.

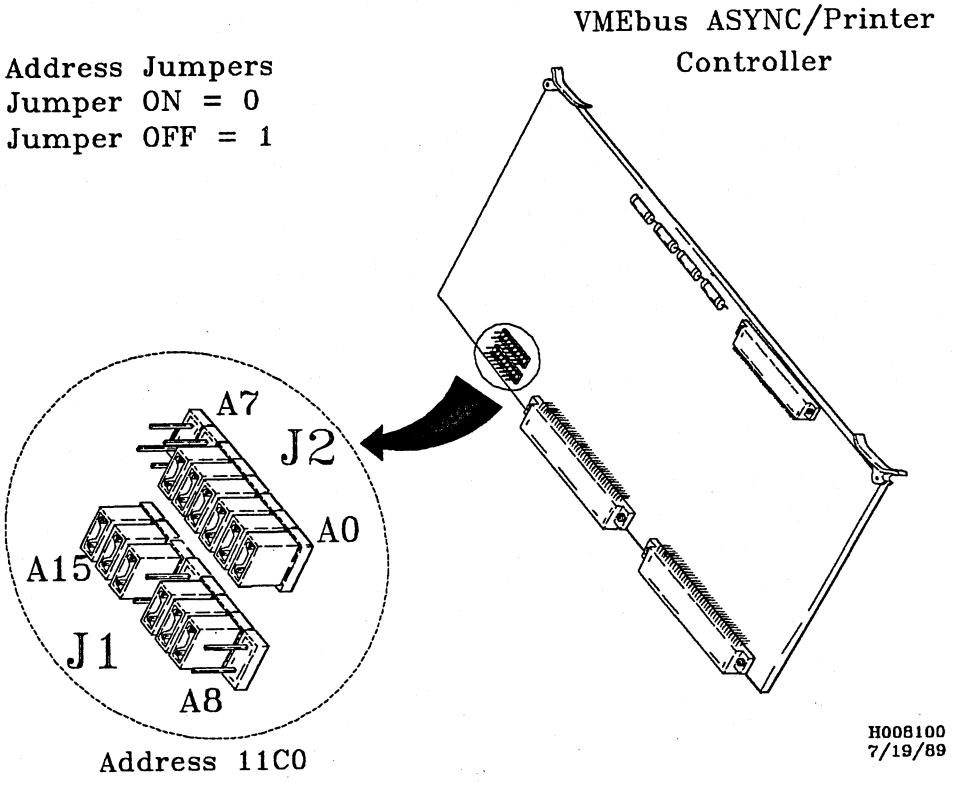
4.6.1.13 Self-Test 13, Octal UART Self-Test

The Octal UARTs are looped back internally and characters written to each transmit register. The self-test verifies that the characters are received as expected in the receive buffer and that the appropriate status is generated. This self-test only tests at the 9,600 baud rate, 8 bits per character with even parity.

4.7 Debug Monitor

The Debug Monitor within the EPROM self-test portion of this diagnostic is designed for use primarily associated with board level debugging. Using the Debug Monitor allows the VIOP and the Service Processor to be bypassed using TTY ports on the VMEbus Async Controller for board level debugging. The console must be attached to a TTY port on the VMEbus Async Controller, the controller address must be set to 0x11c0 (see the following figure), and the VMEbus chassis must be reset to invoke the Debug Monitor.

Figure 4–6, Jumper Settings for Debug Monitor



Upon first entering the debug monitor, the EPROM initializes the DUSCC and UART devices for 9,600 baud, 8 bits per character, 1 stop bit, and no parity. The debugger then checks each port looking for a carriage return (RETURN) at one of the TTY ports. As soon as a carriage return (RETURN) is received, that port address is used for all TTY communication until the controller is reset.

4.7.1 Debug Monitor Commands

The following screen shows all Debug Monitor commands. All numbers are hexadecimal.

Figure 4–7, Debug Monitor Commands

```
d [-bwl] a1 [a2]          - display/modify contents of locations a1 - a2
f [-bwl] a1 [a2] pattern  - fill locations a1 - a2 with pattern
s [-bwl] a1 a2 pattern    - search locations a1 - a2 for pattern
c [-bwl] a1 a2 a3         - copy block of memory a1 - a2 to address a3
r [number [number]]       - run one or more self-tests
p [-acd] [-rlt<number>]   - print test sequence to printer; -a: add <CR>,
  -c: centronics, -d: data - -r: chars/line, -l: lines/page, -t: total chars
q (if in dump/modify mode) - quit dumping/modifying
h                         - display command summary
?                         - display command summary
```

4.7.2 d

Usage: **d** [-bwl] **a1** [a2]

Displays and/or modifies contents of locations a1–a2 where:

-b is for byte access

-w is for word (2 bytes) access

-l is for longword (4 bytes) access (default)

If the ending address (a2) is omitted, this command enters an interactive mode that allows modification of memory. The following list gives all valid responses, while in interactive mode:

[<value>]	write optional <value> to current address; stay at thisaddress
.	stay at the present address (re-read)
-	back up to the previous address
RETURN	advance to the next address
q	write optional <value> to current address; exit interactive mode.

4.7.3 f

Usage: **f** [-bwl] **a1** [a2] *pattern*

Fills locations a1–a2 with *pattern*. If the ending address (a2) is omitted, only the first address (a1) is written. The options are:

-b is for byte access

-w is for word (2 bytes) access

-l is for longword (4 bytes) access (default)

4.7.4 s

Usage: **s** [-bwl] **a1 a2** *pattern*

Searches locations a1–a2 for *pattern*. Each time the *pattern* is matched, the address and data are displayed. The available options are:

-b is for byte access

-w is for word (2 bytes) access

-l is for longword (4 bytes) access (default)

4.7.5 c

Usage: **c** [-bwl] **a1 a2 a3**

Copy a block of memory from a1 to a2 to address a3 where:

-b is for byte access

-w is for word (2 bytes) access

-l is for longword (4 bytes) access (default)

4.7.6 r

Usage: **r** [number [number]]

Runs one or more self-tests. If **r** is entered with no arguments, the test list is displayed. Any number of tests may be specified in any order (separated by a space) but the input line may not be more than 80 characters long.

4.7.7 p

Usage: **p** [-acd] [-rlt<number>]

Prints test sequence to printer. The test sequence is all printable characters from 0x21 to 0x7e, repeated as needed. A line feed (0x0a) is inserted at the end of each line. A carriage return (0x0d) is then inserted if the -a option was invoked. A form feed (0x0e) is inserted after the specified number of lines have been printed. The available options are:

-a add carriage return (0x0d)

-c Centronics printer (default)

-d Data Products

-r number of characters per line (default is 80)

-l number of lines per page (default is 60)

-t total

4.7.8 h or ?

Usage: **h** or: Usage: **?**

Displays command summary.

4.8 Class Descriptions

The *dev5300* test contains the three classes of subtests listed in the following table:

Table 4–3, *dev5300* Test Classes

CLASS	DESCRIPTION
1	Basic Controller Tests
2	Download and Execute Firmware on the Controller
3	Printer FIFO and Interface Tests

All subtests contained in *dev5300* are loopable under the *dshell*.

4.9 Class 1 Subtests

Class 1 subtests consist of basic controller tests involving EPROM, parity, interrupt, and RAM. They test the controller without downloading any code to be executed on the controller. This class of subtests verifies the ability of the VIOP to access the controller, the functionality of the controller map registers, and communicates with the controller EPROM code to run each of the self-tests individually.

Table 4–4, Class 1 Subtests

SUBTEST	DESCRIPTION	TIME (min:sec)
100	VMEbus Async Access Controller Test	00:01
101	VMEbus Async Supervisor Registers Test	00:01
102	VMEbus Async Local Map Registers Test	00:01
103	VMEbus Async VMEbus Address Match Test	00:01
104	VMEbus Async Instruction Memory Self-Test	00:13
105	VMEbus Async Memory Address Self-Test	00:08
106	VMEbus Async DPR Memory Self-Test	00:13
107	VMEbus Async Array Register Self-Test	00:08
108	VMEbus Async Interrupt Self-Test	00:08
109	VMEbus Async Interrupt Self-Test	00:08
110	VMEbus Async Interrupt Self-Test	00:08
111	VMEbus Async Timer Self-Test	00:08
112	VMEbus Async Timer Counter Self-Test	00:08
113	VMEbus Async Timer Scale Self-Test	00:08
114	VMEbus Async Printer FIFO Self-Test	00:08

4.9.1 Subtest 100, VMEbus Async Access Controller Test

This subtest attempts to read the controller reset register at the address specified by the user-selected entry in the *ioconfig* file or the user-specified address/VMEbus.

4.9.2 Subtest 101, VMEbus Async Supervisor Registers Test

This subtest will walk a one then a zero across all valid bits in all registers (which do not cause interrupts or a reset to occur) of the controller in the short address space of the VIOP. These registers and their addresses are:

- MSB of reset register (csr + 1)
- LSB of reset register (csr + 3)
- MSB of local map register (csr + 5)
- MID of local map register (csr + 7)
- LSB of local map register (csr + 9)
- MSB of VMEbus address match register (csr + 11)
- MID of VMEbus address match register (csr + 13)
- LSB of VMEbus address match register (csr + 15)
- VMEbus address modifier match register (csr + 17)

The subtest verifies register independence as well as the integrity of each register.

4.9.3 Subtest 102, VMEbus Async Local Map Registers Test

This subtest verifies the functionality of the map registers. A one is walked across the local map registers with mapping enabled and unique data is written and read to each valid address generated in this manner. After all valid addresses have been written in this method, all of the above addresses are generated again and the data is read back to verify access independence.

4.9.4 Subtest 103, VMEbus Async Address Match Test

This subtest verifies the functionality of the VMEbus address match registers. Data is written, after setting up the map registers, to a location in DPR. A one is walked across the address match registers with mapping enabled and the ability to read the data at each new valid VMEbus address generated by the walking one is verified.

4.9.5 Subtest 104, VMEbus Async Instruction Memory Self-Test

Each byte of the instruction RAM (address 0x0–0x3ffff) is accessed and unique data is written. The memory is then read and verified. Next, the memory has a pattern (0xaaaaaaaa/0x55555555) written and verified throughout memory. The pattern is then inverted and the algorithm repeated.

4.9.6 Subtest 105 VMEbus Async Memory Address Self-Test

This test is a quick verification of longword accessibility in DPR (address 0x40000000–0x4003ffff). A one is walked across the address and unique data is written and verified in DPR.

4.9.7 Subtest 106, VMEbus Async DPR Memory Self-Test

Each byte of DPR (address 0x40000000–0x4003ffff) is accessed and unique data is written. The memory is then read and verified. Next the memory has a pattern (0xaaaaaaaa/0x55555555) written and verified throughout memory. The pattern is then inverted and the algorithm repeated.

4.9.8 Subtest 107, VMEbus Async Array Register Self-Test

A walking ones/zeros test is performed on all valid bits of all read/write registers on the board. The registers to be tested are:

- intr #0 priority
- intr #1 priority
- intr #2 priority
- intr #3 priority
- intr #4 priority
- intr #5 priority
- intr #6 priority
- intr #7 priority
- intr #8 priority
- intr #9 priority
- intr #10 priority
- intr #11 priority
- intr #12 priority
- intr #13 priority
- intr #14 priority
- intr #15 priority
- intr #16 priority
- intr #17 priority
- intr #18 priority
- intr #19 priority
- intr #20 priority
- intr #21 priority
- intr #22 priority
- intr #23 priority
- intr #24 priority
- intr #25 priority
- intr #26 priority
- intr #27 priority
- intr #28 priority
- intr #29 priority
- intr #30 priority
- IER msb
- IER lsb
- FIR msb
- FIR lsb
- timer a base count
- timer a contol reg
- timer b base count
- timer b control reg
- vmebus map reg (msg)
- vmebus map reg (lsb)
- timeout reg
- supervisory base addr
- vmebus intr cntl reg
- vme dma addr modifier
- vme dma base addr reg msb
- vme dma base addr reg lsb
- vme dma local bar msb
- vme dma local bar lsb
- vme dma byte count
- vme dma throttle reg
- io dma 0 addr reg msb
- io dma 0 addr reg lsb
- io dma 0 byte count
- io dma 1 addr reg msb
- io dma 1 addr reg msb
- io dma 1 byte count
- io dma 2 addr reg msb
- io dma 1 byte count
- io dma 2 addr reg lsb
- io dma 3 addr reg msb
- io dma 3 addr reg lsb
- io dma 3 byte count
- DSR

4.9.9 Subtest 108, VMEbus Async Interrupt Self-Test

The interrupt arbitration subtest verifies the ability of the board to schedule multiple simultaneous interrupts of the same priority. The subtest causes two interrupts of each priority to be requested simultaneously using the Force Interrupt Register (FIR). The subtest verifies that these interrupts are scheduled, according to priority, in a round-robin manner for interrupt levels zero through seven.

4.9.10 Subtest 109, VMEbus Async Interrupt Self-Test

Subtest 109 sets up the interrupt priority registers, the Interrupt Vector Base Register, and the Interrupt Enable Register (IER). The subtest enables all interrupts in the IER then causes each interrupt using the Force Interrupt Register (FIR). Each interrupt is caused twice successively and verified in the Interrupt Status Register (ISR).

4.9.11 Subtest 110, VMEbus Async Interrupt Self-Test

Subtest 110 enables all possible 32 interrupts in the Interrupt Enable Register (IER) that are then caused by writing the appropriate bit in the FIR. The interrupt vector that is generated is written to memory to be verified.

4.9.12 Subtest 111, VMEbus Async Timer Self-Test

The functionality of the Timer Control Register for the two timer circuits is verified. All modes of the timers and the proper generation of interrupts are also verified for both timers.

4.9.13 Subtest 112, VMEbus Async Timer Counter Self-Test

A one is walked across the Timer Base Count Register. Also, the ability of the Present Count Register to properly decrement the timer count value is verified.

4.9.14 Subtest 113, VMEbus Async Timer Scale Self-Test

Subtest 113 verifies the timer's pre-scaler functionality by counting the number of iterations of a timing loop with scale set at divide by one (this is used as a reference count.) The count used in the scale of one is multiplied by the inverse of the scale factor (i.e., if the scale is ½ the count is multiplied by 2, if the scale ¼ the count is multiplied by four). The number of iterations of the timing loop with the adjusted count and scale is compared to the base count obtained with the initial scale.

4.9.15 Subtest 114, VMEbus Async Printer FIFO Self-Test

The printer FIFO (1-Kbyte FIFO, 8 bits wide) is filled and then read. The functionality of the FIFO full and FIFO empty status bits, as well as the data integrity is verified. The FIFO empty printer interrupt is also verified.

4.10 Class 2 Subtests

The class 2 subtests first download a firmware module to execute on the controller. This class of tests then communicates with the downloaded module on the controller to perform the following tests:

Table 4–5, Class 2 Subtests

SUBTEST	DESCRIPTION	TIME (min:sec)
200	VMEbus Async Download Test	00:04
201	VMEbus Async Access Test	00:02
202	VMEbus Async Interrupt Test	00:02
203	VMEbus Async Errors Test	00:02
204	VMEbus Async Basic DMA Test	01:24
205	VMEbus Async DMA Counter Test	00:02
206	VMEbus Async DMA Suspend Test	00:02
207	VMEbus Async DMA Throttle Test	00:02
208	VMEbus Async DMA Error Test	00:02
209	VMEbus Async Octal UART Internal Loopback Test	13:20
210	VMEbus Async DUSCC Internal Loopback Test	12:50
211	VMEbus Async Octal UART External Loopback Test	13:20
212	VMEbus Async DUSCC External Loopback Test	12:50

4.10.1 Subtest 200, VMEbus Async Download Test

This subtest verifies the ability to execute a downloaded module in the controller. The code to be executed on the controller is written into the instruction RAM via the map registers, then read back to verify the download. The controller is then reset with the EPROM mapped out. The downloaded code (which starts at address 0) then begins execution. The clearing of the "SYSFAIL" bit in the controller reset register indicates a successful boot.

4.10.2 Subtest 201 VMEbus Async Access Test

This subtest verifies that the 68020 on the controller has the ability to perform VMEbus accesses to and from a VIOP window register. The subtest verifies reads and writes of 1 byte to 11 bytes to addresses with offsets of 0 byte to 3 bytes.

4.10.3 Subtest 202, VMEbus Async Interrupt Test

This subtest verifies the ability of the controller to properly generate VMEbus interrupts as well as receive interrupts. The controller interrupt, acknowledge interrupt, is also verified. The subtest verifies the ability of the controller to recognize the "mailbox" interrupts. The VIOP

generates these interrupts by writing a specific bit in the reset register of the controller. The ability of the controller to take these one at a time as well as simultaneously is verified.

4.10.4 Subtest 203, VMEbus Async Errors Test

This subtest verifies the ability of the controller to recognize the various bus timeout conditions. There are three types of timeouts: VMEbus arbitration, VMEbus dtack, and local bus. The subtest verifies that the board signals bus error for all required conditions as well as verifying that the bus error source register is correct and the bus error exception routine is properly invoked.

4.10.5 Subtest 204, VMEbus Async Basic DMA Test

This subtest verifies the ability of the controller to perform DMA accesses to and from the VIOP. This subtest verifies normal DMA operation of reads and writes of transfers of 1 bytes to 12 bytes at addresses with offsets of 0 byte to 3 bytes.

4.10.6 Subtest 205, VMEbus Async DMA Counter Test

This subtest verifies the counting logic of the VMEbus DMA Controller counter. A one is walked across the DMA counter register and the DMA transfer is started. The transfer is then stopped and the Present Count Register is examined to verify that the count has decremented properly.

4.10.7 Subtest 206, VMEbus Async DMA Suspend Test

This subtest verifies that the DMA Controller suspends a DMA transfer upon assertion of the suspend bit. A transfer is started then suspended. After waiting, the Current Byte Count Register is examined to verify that the transfer was suspended.

4.10.8 Subtest 207, VMEbus Async DMA Throttle Test

This subtest validates the ability of the controller to throttle DMA operations by setting the Throttle Register to four. The DMA transfer is then started normally. After starting the DMA transfer, the 68020 immediately attempts to read the VMEbus address of the starting DMA transfer address plus four times the transfer size. The DMA gives up the bus after four accesses and the 68020 is able to complete its read. Since this memory was previously cleared the 68020 reads a zero. Successive reads to this address results in the DMA (non zero) data being read.

4.10.9 Subtest 208 VMEbus Async DMA Error Test

This subtest verifies the ability of the DMA Controller to recognize error conditions and respond to them properly. All conditions causing a configuration error, a count error, a bus arbitration error, and a bus error are tested.

4.10.10 Subtest 209, VMEbus Async Octal UART Internal Loopback Test

This subtest verifies the functionality of the octal UART in local loopback mode. The subtest uses all baud rates supported by the devices, only one stop bit per character. Both 7-bit and 8-bit characters are used with even, odd, and no parity generated for each character. The baud rates tested are selectable in the test parameter menu that is presented after test invocation. Since synchronous communication requires panel changes (which is not currently supported) only asynchronous communication is tested.

4.10.11 Subtest 210, VMEbus Async DUSCC Internal Loopback Test

This subtest verifies the functionality of the DUSCC devices in local loopback mode. The subtest uses all baud rates supported by the devices, only one stop bit per character. Both 7-bit and 8-bit characters are used with even, odd, and no parity generated for each character. The baud rates tested are selectable in the test parameter menu that is presented after test invocation. Since synchronous communication requires panel changes (which is not currently supported) only asynchronous communication is tested.

4.10.12 Subtest 211, VMEbus Async Octal UART External Loopback Test

This subtest verifies the octal UART device with loopback cables attached. This subtest uses the same algorithm as Subtest 210. Again, only asynchronous communication is tested. Data Terminal Ready (DTR) and Data Send Ready (DSR) loopback are tested in this subtest as well.

4.10.13 Subtest 212, VMEbus Async DUSCC External Loopback Test

This subtest verifies the DUSCC devices with loopback cables attached. This subtest uses the same algorithm as the Subtest 210. Again, only asynchronous communication is tested. Data Terminal Ready (DTR) and Data Send Ready (DSR) loopback are tested in this subtest as well.

4.11 Class 3 Subtest

The purpose of the class 3 test is to test the printer FIFO and interface. The same module that is downloaded for the class 2 tests is also downloaded for this test. Subtest 300 is the only test in this class, as indicated in the following table:

Table 4–6, Class 3 Subtest

SUBTEST	DESCRIPTION	TIME (min:sec)
300	VMEbus Async Printer Test	00:16

4.11.1 Subtest 300, VMEbus Async Printer Test

This subtest verifies the printer subsystem. In order to properly complete this subtest, a printer must be attached to the console. The subtest allows the specification of a file containing the pattern to print, the number of characters per line, the number of lines per page, and the number of pages to print. If the printer is not entered in the ioconfig file, a warning is printed to the terminal and the subtest is not executed.

4.12 Interactive Debugger

The diagnostic provides an interactive debugger that supports the ability to execute commands from a script file. This allows more debug flexibility. Invocation of the interactive debugger is achieved by using the **-d** option when invoking the diagnostic (enter **dev5300**[x] **-d**). Also, the interactive debugger can be invoked by entering **y** to the "Enable Debug Monitor?" prompt. No subtests are executed and the debugger is invoked.

Once the interactive debugger is invoked, online help commands are available. By entering **help** (RETURN), the following screen is displayed:

Figure 4–8, Interactive Debugger Online Help

```
Input base specification:
    0dNN - decimal,  0xNN or NN - hexadecimal,  the default is hexadecimal

Meta-command sequences:
   ![UNIX_CMD]       - execute UNIX_CMD
   !![UNIX_CMD]      - fork a shell and execute UNIX_CMD (allows redirection)
   <FILE             - redirect input from FILE (recursive)
   <<FILE            - end input from current file and change input to FILE

Commands:
    Commands may be abbreviated as long as the abbreviation is unique.

    help    [COMMAND ...]                    - display general or specific help
    cd      [DIRECTORY]                      - change to DIRECTORY
    quit                                     - exit debug mode
    pause   [-n] [seconds]                   - pause for <C/R> or seconds
    mb      begin [end] [step]               - modify/[dump] bytes on CCU
    mw      begin [end] [step]               - modify/[dump] words on CCU
    ml      begin [end] [step]               - modify/[dump] longs on CCU
    mmb     begin [end] [step]               - modify/[dump] bytes in MM
    mmw     begin [end] [step]               - modify/[dump] words in MM
    mml     begin [end] [step]               - modify/[dump] longs in MM
    fb      begin [end] value [incr [step]] - fill bytes on CCU
    fw      begin [end] value [incr [step]] - fill words on CCU
    fl      begin [end] value [incr [step]] - fill longs on CCU
    reset   [0,1]                            - reset Vasync controller
    map     vme local                        - map vme address to Vasync
    load    file                             - download file to Vasync
    csrreg                                   - dump controller's VME regs
    vdma                                     - dump controller's VME DMA regs
    iodma    [0-3]                           - dump controller's I/O DMA regs
```

In addition to the help screen in the previous figure, help for each specific command is obtained by entering:

help *command*

In the previous sequence, replace *command* with the desired debugger command. Abbreviations of the desired commands may be used. For example, for help with all commands that start with the letter "r," enter:

help r

4.13 Interactive Debugger Command Descriptions

4.13.1 help

Usage: **help** [COMMAND ...]

Displays general or specific help, where COMMAND is replaced with the desired interactive debugger command. Specific help is displayed for COMMAND. The COMMAND may be an abbreviation. The following example would list help for all commands that start with the letter "r":

help r

4.13.2 cd

Usage: **cd** [PATH]

Changes to desired directory, where PATH may be any valid directory path. If PATH is omitted, the default path is $HOME or / if $HOME not set.

4.13.3 pause

Usage: **pause** [-n] [seconds]

Wait for specified amount of time or for a RETURN if the time is omitted, where:

-n means do not echo the pause message

seconds specifies the number of seconds to pause

4.13.4 mb, mw, ml

Usage: **mb begin** [end] [step]
mw begin [end] [step]
ml begin [end] [step]

Displays or modifies, or both, CCU address space in byte-at-a-time mode (mb), word-at-a-time mode (mw), or long-word-at-a-time mode (ml), where:

- **begin** is the initial address
- **end** is the ending address (optional)
- **step** is the address increment (optional (default is access size))

If the ending address is omitted, this command enters an interactive mode that allows modification of memory. The following list gives the valid responses while in interactive mode:

[<value>]	write optional <value> to current address, advance to next address
[<value>]=	write optional <value> to current address, and stay at the present address (re-read)
[<value>]^[N]	write optional <value> to current address, move to address N (address 0 if N is omitted)
[<value>]+[N]	write optional <value> to current address, advance to the next address (N addresses if N is specified)
[<value>]-[N]	write optional <value> to current address, back up to the previous address (N addresses if N is specified)
[<value>]q	write optional <value> to current address, exit interactive mode

Multiple commands may be specified on the same line. A comma or space may be used to separate the commands or value as shown in the following example:

```
Debug mode -> mb c03fc1
<CCU:c03fc1> = 1c 00=ff,1q
```

Where: **1c 00=ff,1q** is an example of executing multiple commands on the same line. This sequence modifies the byte at address 0xc03fc1 to 0, re-reads and displays the new value, modifies the byte to 0xff, skips to address 0xc03fc2 and modifies it to a 0x1, and then quits interactive mode.

4.13.5 mmb, mmw, mml

Usage: **mmb begin** [end] [step]
mmw begin [end] [step]
mml begin [end] [step]

Displays or modifies (or both) main memory address space in byte-at-a-time mode (mmb), word-at-a-time mode (mmw), or long-word-at-a-time mode (mml), where:

- **begin** is the initial address
- **end** is the ending address (optional)
- **step** is the address increment (optional (default is access size))

If the ending address is omitted, this command enters an interactive mode that allows modification of memory. The following list gives all valid responses while in interactive mode:

[<value>]	write optional <value> to current address, advance to next address
[<value>]=	write optional <value> to current address, and stay at the present address (re-read)
[<value>]^[N]	write optional <value> to current address, move to address N (address 0 if N is omitted)
[<value>]+[N]	write optional <value> to current address, advance to the next address (N addresses if N is specified)
[<value>]-[N]	write optional <value> to current address, back up to the previous address (N addresses if N is specified)
[<value>]q	write optional <value> to current address, exit interactive mode

Multiple commands may be specified on the same line. A comma or space may be used to separate the commands or values as shown in the following example:

```
Debug mode -> mmb c03fc1
<Main-Mem:c03fc1> = 1c 00=ff,1q
```

Where: **1c 00=ff,1q** is an example of executing multiple commands on the same line. This sequence modifies the byte at main memory address 0xc03fc1 to 0, re-reads and displays the new value, modifies the byte to 0xff, skips to address 0xc03fc2 and modifies it to a 0x1, and then quits interactive mode.

4.13.6 fb, fw, fl

Usage: **fb begin** [end] **value** [incr [step]]
fw begin [end] **value** [incr [step]]
fl begin [end] **value** [incr [step]]

Fills memory with specified pattern in byte-at-a-time mode (fb), word-at-a-time mode (fw), or longword-at-a-time move (fl), where:

- **begin** is the starting address
- **end** is the ending address
- **value** is the initial fill value
- **incr** is the fill value increment
- **step** is the address increment

4.13.7 reset

Usage: **reset** [0,1]

Resets the VMEbus Async Controller by writing (ORing) a one into the Reset Register at address CSR + 1. Then releases reset by ANDing with 0xfe. If reset is issued with a 0, EPROM is mapped out. If reset is issued with a 1, EPROM is mapped in.

4.13.8 echo

Usage: **echo** [-n] [arg ...]

Writes arguments separated by blanks and terminated by a newline to the display, where:

-n does not echo the terminating newline character

4.13.9 quit

Usage: **quit**

Typing **quit** exits the interactive debugger.

4.13.10 map

Usage: **map vme_address local_address**

This command sets up the VMEbus Async Controller to map 4 Kbytes of its internal memory space starting at (local_address & 0xfffff000) to a vme_address. The following list shows what chassis goes with what controller and what address modifier is associated with these addresses:

Addresses 0x1000000 to 0x13fffff are chassis 0, address modifier 0x3d

Addresses 0x1400000 to 0x17fffff are chassis 1, address modifier 0x3d

Addresses 0x1800000 to 0x1bfffff are chassis 0, address modifier 0x0d

Addresses 0x1c00000 to 0x1ffffff are chassis 1, address modifier 0x0d

4.13.11 csrreg

Usage: **csrreg**

Dumps the registers visible in short VMEbus address space. These are the reset, the local map, VMEbus address map, and address map registers.

4.13.12 vdma

Usage: **vdma**

Dumps the controller VMEbus DMA registers. These are the control, status, count, local base address, VMEbus base address, throttle, and current count registers.

4.13.13 iodma

Usage: **iodma** [0 - 3]

Dumps the controller I/O DMA registers. These are the control, status, count, memory base address, and current count registers.

4.14 Error Messages

Two basic types of error conditions can occur when executing *dev5800*. If an error occurs during the controller EPROM self-tests (i.e., during a reset to the controller or when power is applied to the controller), no error messages are output to the console. The self-test error code is displayed

on the controller LEDs and is also stored in the LED Display Register (refer to the "VMEbus Async EPROM Self-Tests" section within this test description for more information).

If an error occurs during execution of *dev5300* that is associated with the diagnostics subtests, errors are output to the console. Each error contains header information with the date and time, and the diagnostic name, class, and subtest. The following is an example of the header information:

Figure 4–9, Error Message Header Format

```
*****  Sun Jul 16 15:04:10 1989  *****
Test:    dev5300.t  1.1    Class: 1    Subtest: 100 1.2   Count: 1    Error: 0
Failed:  VMEbus Vasync Access Controller Test
```

Also, at the end or each error message, a footer is printed containing the diagnostics name and the elapsed time. The following is an example of a footer:

Figure 4–10, Error Message Footer Example

```
Test 'dev5300.t' failed
Elapsed time:   0:00:03
```

The following are the different error messages that are printed with the header and footer information:

```
Error #0x1: Invalid command
Invalid Command: 0x1

Error #0x2: VIOP Fatal Exception Error

Error #0x3: Data Miscompare
Expected: 0x0   Actual 0x1   at address 0x0

Error #0x4: VIOP Window Allocation Error

Error #0x5: VIOP Checksum Error
Expected: 0x0   Actual 0x1
Main Memory Address: 0x0  VIOP Window Address: 0x0

Error #0x6: VIOP Download Error
Read 0x0 from 0x0 in Main Memory thru 0x0
Read 0x1 from 0x0 in Vasync at mapped address 0x0
```

```
Error #0x7: Vasync Reset/Boot Error
Controller did not clear SYS_FAIL bit after reset
Expected: 0x00   Actual 0x01   in CSR + 1

Error #0x8: Vasync Controller Timeout
Controller Timed Out

Error #0x9: Vasync Device Status
Expected: 0x00   Actual 0x01   at address 0x0

Error #0x10: Vasync Timer Counter
Timer Counter Register Address: 0x0
Expected: 0x0000   Actual 0x0001

Error #0xa: Vasync Device Status
Expected: 0x0000   Actual 0x0001   at address 0x0

Error #0xb: Vasync Data Miscompare
Expected: 0x00   Actual 0x01   at address 0x0

Error #0xc: Vasync Data Miscompare
Expected: 0x0000   Actual 0x0001   at address 0x0

Error #0xd: Vasync Data Miscompare
Expected: 0x00000000   Actual 0x00000001   at address 0x0

Error #0xe: Vasync Interrupt Status
Expected Interrupt Status: 0x00000000  Actual   Interrupt Status: 0x00000001

Error #0xf: Vasync Interrupt Count
Expected Interrupt #: 0x00000000  Actual   Interrupt #: 0x00000001

Error #0x11: Invalid command
Invalid Command: 0x1

Error #0x12: Vasync Interrupt Enable
Expected IER: 0x00000000  Actual   IER: 0x00000001

Error #0x13: Vasync VME DMA Throttle
Controller failed to throttle properly
Expected: 0000
Actual  : 0001

Error #0x14: Vasync Local Bus Error
Unexpected Bus Error attempting to read address 0
Expected Bus Error: 0000
Actual Bus Error  : 0001

Unknown Error Number: 0x15
```

THIS PAGE INTENTIONALLY LEFT BLANK

Appendix A
Reporting Problems

A.1 Overview

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

A.2 Technical Assistance Center

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

A.3 The *contact* Utility

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

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

A.4 Prerequisites

To use *contact* requires

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

A.4.1 UUCP Connection

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

A.4.2 Finding the Program Path Name

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

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

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

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

A.4.3 Finding the Program Version Number

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

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

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

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

A.5 Tips on Using the *contact* Utility

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

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

A.5.1 Using a *.contact* File

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

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

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

A.5.2 Aborting the Report

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

A.5.3 Submitting the *dead.report* File

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

contact -r

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

A.5.4 Suspending a Report

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

A.5.5 Ending a Response

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

A.5.6 Tilde-Escape Sequences

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

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

A.6 Using the *contact* Utility

The *contact* utility prompts for the following information:

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

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

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

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

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

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

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

Enter the name of the product involved
>
```

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

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

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

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

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

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

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

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

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

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

Review	Review the text of your contact report. You are then prompted again to select an option.
Edit	Edit the text of the contact report. If you choose to edit the report, *contact* puts you in your default text editor.
Submit	Send the report to the CONVEX TAC. You are notified within 48 hours that the TAC has received the report. This option exits the *contact* utility and returns you to the shell environment.
Abort	Save the text of your report in a file named *dead.report* in your home directory. This option exits the *contact* utility and returns you to the shell environment.

Index

S

T

U

V

W

CONVEX VMEbus Async Controller
(*dev5300*) Diagnostics Manual
Document No. 760-003230-000
First Edition

Reader's Forum

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

From:

Name_____ Title_____

Company_____ Date_____

Address and Phone No._____

FOR ADDITIONAL INFORMATION OR DOCUMENTATION:

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

Direct mail orders to:

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

(Fold Here First)

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 1046 RICHARDSON, TEXAS

POSTAGE WILL BE PAID BY ADDRESSEE

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

(Fold Here Second)

(Tape or Staple)