Extended ROM Resident Diagnostics

SOLBOURNE COMPUTER, Inc.
1900 Pike Road
Longmont, Colorado 80501
(303) 772-3400

Part Number: 101489-AB

July 1989

Preface

This manual describes **rdg**, the Solbourne Computer, Inc., ROM-resident diagnostics program. This manual is divided into four sections and an appendix, as follows:

Section 1 - Introduction
This section introduces the Bootable/Standalone Diagnostics program **rdg**.

Section 2 - Getting Started with **rdg**
This section explains how to begin using **rdg**.

Section 3 - **rdg** Tests
This section presents the tests currently available using **rdg**.

Section 4 - Commands
This section gives the user commands available when using **rdg**.

Appendix A - MARCH Algorithm
This appendix gives an explaination of how the MARCH algorithm works.

Appendix B - Moving Inversions Test Algorithm
This appendix gives an explaination of how the moving inversions algorithm works.

Table of Contents

LIST OF FIGURES

Section 1: Introduction

1.1 Introduction

rdg is a ROM-resident diagnostics program. It is used to determine why a Solbourne system will not boot, if problems are encountered while booting the system. This program is for use primarily by engineering personnel.

The software for **rdg** includes:

- the **rdg** ROM-resident diagnostics program (**rdg** (1))
- test control commands
- **rdg** tests

1.2 Related Documentation

Information that may be useful while using the **rdg** program is available in the following documentation:

- *Series4/600 Service Manual*, Part number 101249-AA
- *Series4/600 Theory Manual*, Part number 101250-AA
- *Series4/500 Service Manual*, Part number 102161-AA
- *Bootable/Standalone Multiprocessor Diagnostics Manual*, Part number 101686-AB
- *Bootable/Standalone Diagnostics Manual*, Part number 101490-AB
- *System Power On Self Test Manual*, Part number 101486-AB

Section 2: Getting Started with rdg

2.1 Introduction

This section gives step-by-step instructions and examples for getting started using **rdg**.

In this section, commands you enter are given in **boldface** type. Command parameters for which you substitute a value are given in *italic*.

2.2 Invoking the rdg Program

The steps to follow the first time **rdg** is invoked are given below.

The user must first bring the Solbourne system to the **ROM>** prompt. If UNIX is running, it must be shutdown using the **halt** (1) command.

1. At the ROM> prompt, type:

   ```
   ROM>rdg
   ```

2. When **rdg** is invoked, the **RDG>** prompt is displayed. There is no startup message.

   When **rdg** is invoked the following values are given:

   - The memory configuration table and memory limits are initialized with the values found by the ROM during the self-test.
   - The frame buffer configuration table is initialized with the values found by the ROM during the self-test and the default frame buffer defaults to the frame buffer in the highest numbered slot.
   - The VMEbus table comes up as empty.

2.3 Entering Commands to the rdg Prompt

rdg accepts input when the

```
RDG>
```

prompt is displayed.

Commands and parameters are case insensitive.

Other rules for entering commands include:

- In general more than one command can be entered in a single command line to the `RDG>` prompt at the same time.

  ```
  RDG> tests 1 2 3 names on passlim 0 between 5 run
  ```

  The above command line selects tests 1, 2, and 3, turns the printing of test names on, sets the pass limit to 0 (no passlim), the between count is set to 5, and begins test execution with the **run** command.

- Commands that process user input in an interactive mode, such as the **vmeconfig** (1) and **fbconfig** (1), cause commands that follow on the command line to be ignored.

- Commands must be separated by white space(s), including tabs or spaces. (Semicolons are not recognized by **rdg** as spaces.)
- If any of the command(s) entered return an error condition, all following commands are ignored and the `RDG>` prompt is redisplayed.
- If a command is unrecognized by **rdg**, the following is displayed:

  ```
  Unknown command (command name)
  ```

- All command lines are terminated by a **Return**.
- Some commands may display additional error messages if numeric values are entered incorrectly or if the numeric values are not legal. These messages identify the value that is out of range, for example

  ```
  illegal address (value given)
  ```

  If an illegal value is given, additional information may be displayed that identifies the legal range of values.
- Memory and I/O addresses and contents must be entered in hexidecimal format. Any value that has to do with hardware must also be entered in hexidecimal (e.g., register data, memory address, or memory data).
- Counters and test numbers should be entered in decimal format (e.g., counts and limits).
- The **rdg help** (1) command can be used any time the `RDG>` prompt is displayed. A summary of the command given as an argument to **help** will be displayed.

2.4 Using rdg Commands

Example usage of each **rdg** command is given in Section 4 of this manual. All commands can be used with any other commands. All the **rdg** commands are for test control.

The test control commands are commands so categorized because they cause execution or alter the execution of the test programs.

2.4.1 Test Control Commands

The test control commands allow users to control tests run by **rdg**. The command names and their functions follow:

- **between** (1) - Set or display between count
- **config** (1) - Display memory configuration file
- **continue** (1) - Set or display continue on error flag
- **errlim** (1) - Set or display error limit
- **errors** (1) - Display error count
- **fbconfig** (1) - Generates (or modifies) the frame buffer configuration file
- **help** (1) - Display a summary of the command given as an argument to **help**
- **limit** (1) - Display or set memory test limits
- **loop** (1) - Set or display loop on test flag

- **menu** (1) - Display listing of available tests
- **names** (1) - Enable or disable printing of test names during test execution
- **next** (1) - Execute next selected test
- **passes** (1) - Display pass count
- **passlim** (1) - Set or display pass limit
- **prompt** (1) - Set or display prompt flags
- **quiet** (1) - Set or display error message enable flag
- **quit** (1) - Exit from **rdg** debugger program
- **restart** (1) - Restart execution of selected tests
- **run** (1) - Start execution of selected tests
- **status** (1) - Display or reset state of modes, flags, and counts
- **tests** (1) - Select or display tests to be executed
- **time** (1) - Set or display time flag and print current time and date
- **vmeconf** (1) - Configure VMEbus devices

2.5 Starting Test Execution

By default, when **rdg** is invoked all the tests are selected. Tests are executed when the **run** (1) command is entered at the command line. For example:

```
RDG> tests run
```

If the **tests** command is entered without an argument, all the selected tests are displayed. For example:

```
RDG> tests
selected tests:   1    2    3    4    5    6    7    8
                  9   10   11   12   13   14   15   16
```

The test selection can be modified at any time the **RDG>** prompt is displayed. For example:

```
RDG> tests 9 8 4
RDG> tests
selected tests:   9   8   4
RDG>
```

The **menu** command identifies the test names or their functions. For example

```
RDG> menu
Menu of installed test programs:
Test 1: RTC-58321 real time clock test
Test 2: Memory Data RAM test (affected by prompt)
Test 3: Memory ECC RAM test (affected by prompt)
Test 4: VMEbus address map RAM test
Test 5: VMEbus data path test
Test 6: VMEbus address path test
Test 7: RF3500 SCSI data path (Write Buffer) test
Test 8: I/O ASIC register access test
Test 9: I/O ASIC FIFO/ECC test
Test 10: 7990 LANCE initialization test
Test 11: 7990 LANCE internal loopback test
Test 12: 7990 LANCE external loopback test (must be prompted)
Test 13: 33C93 SBIC (SCSI) enable test
Test 14: 33C93 SBIC (SCSI) data path (Write Buffer) test
Test 15: Ethernet tftp read test
Test 16: Disk write/read test
Test 17: Tape write/read test (must be prompted)
RDG>
```

For additional information on test execution, see the **tests** (1), **run** (1), and **menu** (1) commands in Section 4.

2.5.1 Variations of Test Execution

This subsection discusses some of the basic variations that can be applied to test commands. There are other variations than those given here.

Two results can occur during test execution. The test can pass or the test can fail.

If the test passes, the user can do any of the following:

- Tell the controller how many iterations to run using the **passlim** (1) command
- Controls whether the test names are printed using the **names** (1) command
- The user can also stop test execution at any time by entering a Control-C (^C)

If the test fails, the user can do any of the following:

- Set up a oscilloscope loop by using the **loop** (1) and **quiet** (1) commands
- Continue the failing test with the **run** command
- Skip to the next test in the selected sequence of tests using the **next** (1) command
- Restart the entire sequence using the **restart** (1) command.

2.6 Handling Test Failures

Several of the commands given in Section 2.4.1 that are used for test control can be used when test failures occur. In the following example, test 1 detects a failure and the **loop** and **quiet** commands are used to set up a scope loop.

```
RDG> tests 2 run
Starting Test 2: Memory Data RAM Test (affected by prompt)

TEST 2 ERROR:  Thu Dec 1 10:03:05 1988
Error occurred in data ram memory test
Error code = 0xe0  Virtual addr = 0x00800000
                   Physical addr = 0x00ea0000   Board slot = 2
A data failure was found in the second read on the reverse pass.
   exp = 0x55555555
   act = 0x5555555d
   xor = 0x00000008

RDG>
```

Note that test 2 has displayed its error message which identified the failing test case and returned to the **RDG>** prompt. If the user wishes to evaluate this test failure by setting up a scope loop, the sequence of commands shown in the following illustration may be entered.

```
RDG> loop on run

TEST 2 ERROR:  Thu Dec 1 10:03:05 1988
Error occurred in data ram memory test
Error code = 0xe0  Virtual addr = 0x00800000
                   Physical addr = 0x00ea0000   Board slot = 2
A data failure was found in the second read on the reverse pass.
   exp = 0x55555555
   act = 0x5555555d
   xor = 0x00000008

RDG>
```

Note that test 1 has repeated the failing test case and has redisplayed the same error message. This suggests the presence of a solid failure. To speed up the loop and avoid having to reenter the **run** command, the sequence of commands in the following illustration may be entered.

```
RDG> quiet on run

(No information is displayed while the test is looping when the quiet command is invoked.)

^C
```

No information is displayed while the **loop** and **quiet** flags are set. The program is not hung. It

is executing in the tightest possible loop of the failing test case. A Control-C must be entered to halt the loop and return to the RDG> prompt.

☆ ☆ ☆ NOTE ☆ ☆ ☆

It is a common mistake to forget to reset the **loop** and **quiet** flags before restarting the test sequence. This causes the first test case in the first test to be executed in a tight loop. See the **loop** and **quiet** commands in Section 4 for additional information.

2.7 Exiting rdg

To exit **rdg** use the **quit**(1) command.

Section 3: rdg Tests

3.1 Introduction

This section explains the functionality of the tests shipped by Solbourne Computer for use with the rdg(1) debugger. These tests include:

1. RTC-58321 Real Time Clock Test
2. Memory Data RAM Test (affected by **prompt**)
3. Memory ECC RAM Test (affected by **prompt**)
4. VMEbus Address Map RAM Test
5. VMEbus Data Path Test
6. VMEbus Address Path Test
7. RF3500 SCSI Data Path (Write Buffer) Test
8. I/O ASIC Register Access Test
9. I/O ASIC FIFO/ECC Test
10. 7990 LANCE Initialization Test
11. 7990 LANCE Internal Loopback Test
12. 7990 LANCE External Loopback Test (must be prompted)
13. 33C93 SBIC (SCSI) Enable Test
14. 33C93 SBIC (SCSI) Data Path (Write Buffer) Test
15. Ethernet tftp Read Test
16. Disk Write/Read Test
17. Tape Write/Read Test (must be prompted)

☆ ☆ ☆ **NOTE** ☆ ☆ ☆

Error messages from one test are not valid, if failures have occurred during previous tests. The errors from a test must be corrected before advancing to the next test.

3.2 RTC-58321 Real Time Clock Test

This test verifies that the real time clock (RTC) internal registers can be accessed and that the clock is counting. The chip used by the system is a RTC-58321. This is a slow bus chip that is located at RIO address 17020000.

This test consists of two parts:

1. Tests the RTC registers by writing test patterns and then reading them back for verification.
2. Tests verifies that writing the RTC chip can count by loading the registers such that a roll-over causes a ripple from the least significant digit to the most significant digit.

Possible error messages follow:

```
RTC register write/read failure
RTC register address = RR
    write pattern = P
    exp = E
    act = A
    xor = X

where:
    RR - RTC register number
    P  - pattern written to RTC register
    E  - value expected to be read from RTC register
    A  - actual value read from RTC register
    X  - xor of expected and actual values
```

This error message implies that there may be a problem accessing the RTC chip from the slow bus.

```
Clock state rollover test failure (24 hr mode)
                    S  MI HR W D  MO YR
   Initial state  = 59 59 b2 6 31 12 99
   Expected state = 04 00 80 0 01 01 00
   Actual state   = AA AA AA A AA AA AA

where:
   AA - actual state of RTC register
```

If this error occurs, an RTC chip failure has occurred.

3.3 Memory Data RAM Test (affected by prompt)

This test is an extended version of the ROM power-up self-test Addressing and Data Test. The test program performs a moving inverse test algorithm to verify the addressing and data paths (see the Appendix B for information on the moving inverse test algorithm).

The test program determines the test area from the Kbus memory limit entry in the memory limit table (refer to the **limit** (1) command to modify the test area).

The total test area is blocked into eight megabytes or less blocks.

During the read-write-read sequence, the target memory block is cached and checked for correct data. The data in the cache is then complemented and the block is flushed back to memory. The target block is then re-read and verified to contain the complemented data.

Legal error codes for the Data RAM Test are:

- 0x00 - Data fault exception occurred during write of memory with initial data pattern
- 0x10 - Data fault exception occurred during flush of a memory block on write of memory with initial data pattern
- 0x20 - Data fault exception occurred on first read of forward pass
- 0x80 - Data miscompare occurred on first read on forward pass
- 0x30 - Data fault exception occurred during flush of target memory block back to memory during forward pass
- 0x40 - Data fault exception occurred on second read of forward pass
- 0xa0 - Data miscompare occurred on second read on forward pass
- 0x50 - Data fault exception occurred on first read of reverse pass
- 0xc0 - Data miscompare occurred on first read on reverse pass
- 0x60 - Data fault exception occurred during flush of target memory block back to memory during reverse pass
- 0x70 - Data fault exception occurred on second read of reverse pass
- 0xe0 - Data miscompare occurred on second read on reverse pass

An example of a Data RAM Test failure follows:

```
Error occurred in data RAM memory test
Error code = 0xe0  Virtual addr = 0x00800000
                     Physical addr = 0x00ea0000   Board slot = 2
A data failure was found in the second read on the reverse pass.
  exp = 0x55555555
  act = 0x5555555d
  xor = 0x00000008
```

3.4 Memory ECC RAM Test (affected by prompt)

The test is an extended version of the ROM power-up self-test Addressing and Data Test. The test program performs a moving inverse test algorithm to verify the addressing and data paths (see Appendix A for information on the moving inverse test algorithm).

The Error Correction Code (ECC) test uses double word stores and loads to/from memory, the data RAM test uses word.

The test program determines the test area from the Kbus memory limit entry in the memory limit table (refer to the **limit** (1) command to modify the test area).

The total test area is blocked into 8 megabyte or less blocks.

During the read-write-read sequence, the target memory block is cached and checked for correct data. The data in the cache is then complemented and the block is flushed back to memory. The target block is then re-read and verified to contain the complemented data.

Legal error codes for the ECC test are:

- 0x00 - Data fault exception occurred during write of memory with initial data pattern
- 0x10 - Data fault exception occurred during flush of a memory block on write of memory with initial data pattern
- 0x20 - Data fault exception occurred on first read of forward pass
- 0x80 - Data miscompare occurred on high word in first read on forward pass
- 0x90 - Data miscompare occurred on low word in first read on forward pass
- 0x30 - Data fault exception occurred during flush of target memory block back to memory during forward pass
- 0x40 - Data fault exception occurred on second read of forward pass
- 0xa0 - Data miscompare occurred on high word in second read on forward pass
- 0xb0 - Data miscompare occurred on low word in second read on forward pass
- 0x50 - Data fault exception occurred on first read of reverse pass
- 0xc0 - Data miscompare occurred on high word in first read on reverse pass
- 0xd0 - Data miscompare occurred on low word in first read on reverse pass
- 0x60 - Data fault exception occurred during flush of target memory block back to memory during reverse pass
- 0x70 - Data fault exception occurred on second read of reverse pass
- 0xe0 - Data miscompare occurred on high word in second read on reverse pass
- 0xf0 - Data miscompare occurred on low word in second read on reverse pass

An example of an ECC test failure follows:

```
Error occurred in ECC RAM memory test
Error code = 0x40  Virtual addr = 0x00800000
                        Physical addr = 0x00ea0000 Board slot = 2

An exception occurred after re-read of a double on the forward pass
Exception type = data fault, FCR = 0x02, Syndrome = 0x00
```

3.5 VMEbus Address Map RAM Test

This test verifies that the VMEbus address map RAM on the System Board is accessible via RIO transactions. It does not verify that Kbus/VMEbus address translations can be performed.

The VMEbus map RAM is a 2048 word memory and is accessible at low address 83000000 and high address 83ffe000. The adjacent locations in the RAM are at 2000 hexadecimal increments.

Strategy: Write random byte sequence to 2048 VMEbus map RAM locations. Read and verify contents of VMEbus map RAM.

Possible error messages follow:

```
VMEbus address map write/read error
   RIO address = 0xYYYYYYYY
   exp  = EEEEEEEE
   act  = AAAAAAAA
   xor  = XXXXXXXX

where:
    YYYYYYYY - RIO address where error occurred
    EEEEEEEE - value expected to be read from address
    AAAAAAAA - actual value read from address
    XXXXXXXX - xor of expected and actual values
```

If this test fails, the control logic for accessing the RAM should be checked to be functional. If the control logic is operational, then insure the RAM chips are receiving address and data correctly.

3.6 VMEbus Data Path Test

This test verifies correct data path access to the VMEbus using RIO cycles. Note that a VMEbus resident RAM Board must be installed and configured using the **vmeconfig** (1) command. This test requires a VMEbus Memory Board installed in the VMEbus backplane and the boards' presence listed in the VMEbus configuration table (see the command 'vmeconfig' for more information).

The test program writes and reads the VMEbus RAM Board as a 16 bit device and an 8 bit device. The 8 bit device test verifies that the correct VMEbus data strobe line is used and that the unused data strobe does not. The test is run for each data strobe line.

```
VMEbus data error, 16 bit accesses
  address = 0x86100000
  act = 0xfffe
  exp = 0xffff
  xor = 0x0001

VMEbus data error, 8 bit accesses
  address = 0x86100000
  act = 0xff02
  exp = 0xff00
  xor = 0x0002
```

3.7 VMEbus Address Path Test

This test verifies that address lines on the VMEbus backplane do not interact with each other. Note that a VMEbus resident RAM Board must be installed and configured using the **vmeconfig** (1) command. This test requires a VMEbus Memory Board installed in the VMEbus backplane and the boards' presence listed in the VMEbus configuration table (see the **vmeconfig** (1) command for more information).

The test writes and reads the VMEbus RAM Board as a 16 bit device. The test program sets the entire VMEbus ram to 0x0000, then each single bit address line is set to 0xffff. For example:

```
address       contents
  0x000000      0x0000
  0x000002      0xffff
  0x000004      0xffff
  0x000006      0x0000
  0x000008      0xffff
  0x00000a      0x0000
  0x00000c      0x0000
  0x00000e      0x0000
  0x000010      0xffff
  0x000012      0x0000
  .             .
  .             .
  0x0fffff      0x0000
```

In this way any address line that interacts with another one will write the opposite data to the location.

The test is repeated using a single low bit in the address.

```
VMEbus addressing error
  address = 0x86100000
  act = 0xffff
  exp = 0x0000
```

3.8 RF3500 SCSI Data Path (Write Buffer) Test

This test verifies functionality of the VMEbus emulator on the System Board and the connection of the SCSI device to the Ciprico RIMFIRE 3500 SCSI (Ciprico) Board.

This test requires a Ciprico VMEbus SCSI Board installed in the VMEbus backplane and the boards' presence listed in the VMEbus configuration table (see the command 'vmeconfig' for more information).

The test sequence follows:

1. The Ciprico SCSI Board is reset and the test waits for the board to complete self tests.
2. The board options are set with a SCSI id of 7.

3. The unit options are set with a retry limit of 3.
4. The test waits for a minimum of 5 seconds to allow the device to recover from the SCSI bus reset.
5. The test performs a SCSI TEST UNIT READY command. This first command will fail with a check condition status and a sense key of 0x06. This error is expected, it is the drive saying it was reset.
6. The size of the devices buffer is queried and is used for the transfer count. This count is displayed.
7. The data to transfer to the device is set up, an incrementing pattern is used.
8. The data is transferred to the device using the SCSI WRITE BUFFER command.
9. The data is transferred from the device using the SCSI READ BUFFER command.
10. The data read is verified to be the same data that was sent.
11. The data transfer, write-read-verify sequence, is repeated when loop is on.

An example of a data failure follows:

```
Data error on SCSI transfer
  transfer offset = 350
  exp = 0x02
  act = 0x03
  xor = 0x01
```

Errors in this test can be from:

- The System Board will not address the Ciprico SCSI (run VMEbus Data and Address path tests to verify).
- The System Board will not allow the Ciprico Board to address system RAM (run System Board VMEbus address map RAM test to verify mapping RAM). This is not the only cause for this condition.
- The 'Error on first selection of device' error specifies that SCSI device address 0 and Logical Unit Number (LUN) 0 could not be selected. Verify the device:
  1. cables are not damaged and connected
  2. is addressed as device 0
  3. is addressed as LUN 0

SCSI errors return the command that was issued and the device that was addressed. The commands used are shown in the following table.

Command	Device	Description
0x00	0x00	SCSI command, TEST UNIT READY
0x07	0xff	Ciprico Board command, set general options
0x07	0xff	Ciprico Board command, set device options
0x3b	0x00	SCSI command, WRITE BUFFER
0x3c	0x00	SCSI command, READ BUFFER

Errors returned from the device will display Ciprico Board error status (error code) and various SCSI status bytes. Refer to the Ciprico RIMFIRE 3500 Product specification and the Small Computer Systems Interface specification for further information.

Example of an error on a command follows:

```
Error status returned after command
  error code = 0x4b SCSI status = 0x02 SCSI flags = 0x00
  info byte 3 = 0x00  4 = 0x00  5 = 0x00  6 = 0x00
Command 0x3c on device 0x00
```

3.9 I/O ASIC Register Access Test

This test verifies that the I/O ASIC will retain data in internal registers and verify the access to the Western Digital (33C93) SCSI Bus Interface Controller (SBIC) and the AMD 7990 Local Area Network Controller for Ethernet (LANCE) chips. Refer to the technical manuals of these devices for additional information.

For each test, the invlid bits are masked before comparing data.

The first part of this test checks all I/O ASIC registers for data retention. Each register has a series of patterns written, read back and verified.

```
Data error in Ethernet receive address register
    exp = 0x00000000
    act = 0x00000040
```

The test continues by verifying that each of the I/O ASIC's five (5) read/write rergisters are unique. The test sequence for this test is as follows:

```
clear all registers
writing the test register to all ones
check all for proper data
write all registers to all ones
clear the test register
check all for proper data
```

The above example is repeated for all registers as the test register.

```
Data retention error in test register
test register = Ethernet receive address register
    exp = 0x0001ffff
    act = 0x0001ffef
```

In the above example, the test registers did not retain the value written to it.

```
Addressing error on ASIC
test register = SCSI DMA address register
value written = 0x00000000
register modified = SCSI control/address register
    exp = 0x00007ffd
    act = 0x00000000
```

In the above example, writing to the SCSI DMA address register caused the SCSI control/address register to be modified.

The next two tests verify accessability and data retention of the AMD 7990 LANCE chip and the WD 33C93 SBIC chip. Each device is presented a pattern and read back. The LANCE has a 16 bit data bus and the 33C93 has an 8 bit.

The LANCE test uses CSR1 for the test.

```
Data retention error in LANCE chip
    exp = 0xfffe
    act = 0xfefe
```

The 33C93 test uses the total cylinders, low byte, register for the test.

```
Data retention error in SCSI chip
    exp = 0x08
    act = 0x00
```

3.10 I/O ASIC FIFO/ECC Test

This test verifies the I/O ASIC will fetch a cache block from KBus memory and that single bit errors in any cacheline will be corrected. Every data bit in every cache line is verified to be correctable. The test verifies the following conditions:

1. A zero bit corrected to a one in a field of zeroes
2. A one bit corrected to a zero in a field of ones
3. A one bit corrected to a zero in a field of zeroes
4. A zero bit corrected to a one in a field of ones

The LANCE chip is used to force the I/O ASIC to fetch the test cache block.

The cache block is written to memory with one cache line with bad data and good ECC. The I/O ASIC is forced to fetch the block and the cache block is read again. The data read is verified to be corrected.

```
I/O ASIC FIFO/ECC error
 check byte = 0x0c
   addr              exp                    act                    xor
0x000c0000    0x0000000000000000    0x0000000080000000    0x0000000080000000(
```

In the above example, the I/O ASIC did not correct the bad bit in the cache line or it did not fetch the block.

3.11 7990 LANCE Initialization Test

This test verifies that the AMD 7990 LANCE chip can initialize through the I/O ASIC from Kbus memory. To verify the initialization, the LANCE status register is verified to contain the expected status.

Ignored bits are masked off.

```
LANCE initialization error
    exp status = 0x0020
    act status = 0x0030
    mode value = 0x0002
```

In the above example, the LANCE chip received the wrong mode value.

Refer to the AMD 7990 technical manual for further information.

3.12 7990 LANCE Internal Loopback Test

This test verifies proper operation of the LANCE with the I/O ASIC, its' address registers and interrupts to the processor. The test initializes the LANCE for internal loopback mode and sets up a transmit packet of 32 bytes, the maximum for the LANCE, and transmits it to itself. The received packet is then verified.

The expected vector for the LANCE chip is 0x87. Interrupts are checked for valid on:

1. Initialization
2. Transmit packet
3. Receive packet

```
Unexpected interrupt active after LANCE initialization
    No interrupts expected
    vector received = 0x80
```

In the above example, a device that is not being tested generated an interrupt when the LANCE did something.

```
Data error on internal LANCE loopback
   addr              exp                    act                    xor
0x000c00c0    0x0001020304050607    0x0001020384050607    0x0000000080000000
```

In the above example, a bad bit was found in the I/O ASICs' Ethernet cacheable block buffer.

3.13 7990 LANCE External Loopback Test (must be prompted)

This test verifies the operation of the LANCE chip with the data encoding component's on the System Board.

An external loopback connector must be installed before this test is run. Figure 3-1 shows the pin layout for the transceiver connector.

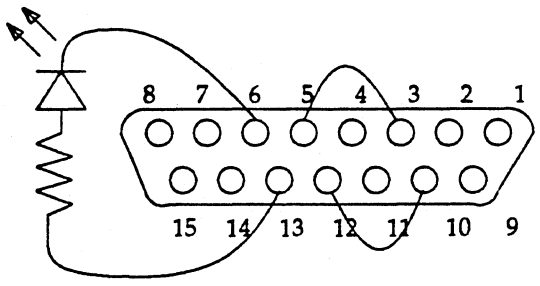

Figure 3-1. Pin Layout for Transceiver Connector on EXOS 202

```
Time out on LANCE transmit
  LANCE status = 0x0033
```

In the above example, the external loopback connector is not connected or an external component is bad.

3.14 33C93 SBIC (SCSI) Enable Test

This test verifies the SCSI chip reset latch will hold the SCSI chip reset and release it from reset. The SCSI chip will not load a value into any register when it is reset. This is the indication that it is reset.

```
SBIC chip not reset after reset latch set
    exp = non zero
    act = 0x00

SBIC chip reset after reset latch reset or data error
    exp = 0x00
    act = 0xff
```

3.15 33C93 SBIC (SCSI) Data Path (Write Buffer) Test

This test verifies the operation of the WD 33C93 SBIC chip (SCSI) with the I/O ASIC.

The chip is set with a SCSI id of 7 (highest priority) and interrupts are verified to occur and be the correct vector.

The test sequence is as follows:

- The test waits for a minimum of 5 seconds to allow the device to recover from the SCSI bus reset.
- The test performs a SCSI TEST UNIT READY command. This first command will fail with a check condition status and a sense key of 0x06. This error is expected and accounted for, it is the drive saying it was reset.
- The data to transfer to the device is set up, an incrementing pattern of 512 bytes is used.
- The data is transferred to the device using the SCSI WRITE BUFFER command.
- The data is transferred from the device using the SCSI READ BUFFER command.
- The data read is verified to be the same data that was sent.

An example of a data failure follows:

```
SCSI transfer error
   addr              exp                    act                    xor
0x000c0000    0x0001020304050607    0x0001020384050607    0x0000000080000000
```

The "Timeout on access to target 0 after reset" error specifes that SCSI device address 0 and Logical Unit Number (LUN) 0 could not be selected. Verify the device:

1. Cables are not damaged and connected
2. Is addressed as device 0
3. Is addressed as LUN 0

SCSI errors return the command that was issued. The commands used are:

```
command   device   description
 0x00      0x00     SCSI command, TEST UNIT READY
 0x3b      0x00     SCSI command, WRITE BUFFER
 0x3c      0x00     SCSI command, READ BUFFER
```

Errors returned from the device will display:

1. SCSI status
2. Target status
3. Command phase
4. SBIC chips status (auxilliary status)

For example:

```
          SCSI     target    command    auxilliary
        status     status     phase       status
exp = 0x02       0x02       0x02         0x02
act = 0x02       0x02       0x02         0x02
```

Please refer to the 33C93 Technical Reference and the the Small Computer Systems Interface (SCSI) specification for further information.

3.16 Ethernet TFTP Read Test

The Ethernet TFTP Read Test verifies the Solbourne workstation's ability to read a file from a host servers's file system.

The file system on the server must have a "/export/exec/kvm/series[45]/stand" directory which contains the "tftpread.data" file. The host server must also be set up to be a tftp file server. This means that the /etc/hosts and /etc/ethers file must contain entries which map the ethernet address of the workstation under test to its internet address.

The "DIAGSERVER" ROM environment variable on the Solbourne workstation under test must be set to accurately point to the directory containing tftpread.data on the server. For example, "DIAGSERVER = tftp.ex(,,1)/export/exec/kvm/series[45]/stand." tftp.ex is the name associated with the EXELAN ethernet device, tftp.ei is the name associated with the IOASIC ethernet device.

If the "DIAGSERVER" ROM environment variable is not set, the test will print:

```
Environment variable DIAGSERVER not initialized
  Test Skipped.
```

When the test attempts to open the file on the remote file system a "rarp" (reverse arp) operation is performed in which the workstation sends out its Ethernet address and the server responds by sending back the internet address of the requesting workstation along with the internet address of the responding server.

If the server fails to respond, a string of messages like the following will be seen as the driver keeps retrying the rarp using its ethernet address. If the server never responds, the rarp request will timeout.

```
rarp: requesting internet address for 8:0:14:20:38:92
rarp: requesting internet address for 8:0:14:20:38:92
rarp: requesting internet address for 8:0:14:20:38:92
rarp: requesting internet address for 8:0:14:20:38:92
rarp: timeout on request
```

If any of the above situations arise, the user should make sure that the server and workstation under test have been setup correctly and that all Ethernet cable connections are secure.

If the server responds normally, several "rarp" messages will be seen on the console:

```
rarp: using IP address 192.9.201.140 = C009C98C
rarp: server at IP address 192.9.201.27 = C009C91B
```

In the above messages, the first message represents IP address of the workstation and the second message represents the IP address of the responding server.

If the initial open of the file pointed to by the "DIAGSERVER" variable fails, the test will print the following message and halt.

```
tftp.ex(,,1)/export/exec/kvm/series[45]/\
                          stand/tftpread.data cannot be opened
```

If the file open is successful, the test reads the data packets sent from the server and compares the data with a re-generated copy of the data. If a read fails, the test will print the following message:

```
Read Error:  tftp.ex(,,1)/export/exec/kvm/series[45]/\
                        stand/tftpread.data, fd = 3, count = 0
```

Where fd is the file descriptor returned from the file open operation and count is the number of bytes returned from the read call. The read operation always requests 4 bytes at a time.

If there are no errors returned from the read operation, the test compares the data read with expected data. If there is a data miscompare, the test will print the following messages and halt:

```
Data Error:  tftp.ex(,,1)/export/exec/kvm/series[45]/\
                                                stand/tftpread.data, fd = 3
byte number 0;
  exp = 0x08a6c2b1
  act = 0xff08a62c
```

A data miscompare error indicates that the data stream was corrupted by the Ethernet device (EXELAN or ASIC) during the time it was received from the network cable and placed in system memory. In any case, the entire data path (cable, Ethernet device and System Board) should be suspect.

3.17 Disk Write/Read Test

The Disk write/read test verifies the ability of the Solbourne workstation to write data to a disk and read it back. The test is device independent in that it will run on any disk device supported by the standalone driver (SCSI or SMD).

The test performs 128 Kbyte writes and reads on the disk partition specified by the DEFAULTSWAP ROM environment variable. The DEFAULTSWAP variable must be valid for the test to operate correctly. In addition, incorrectly setting the DEFAULTSWAP device could result in corruption of disk data.

If the DEFAULTSWAP variable has not been initialized, the test will print the following message:

```
Environment variable DEFAULTSWAP not initialized
    Test Skipped.
```

If the DEFAULTSWAP variable has been initialized. The test attempts to open the specified device. If the open fails for any reason, the test will print the following message and halt (assuming DEFAULTSWAP is set to **sd.si(,,1)**:

```
Default swap device sd.si(,,1) cannot be opened
```

Once the specified disk device has been successfully opened, the test calls a driver function which returns the number of disk blocks associated with the specified partition. If the specified partition is of zero size, the test prints the following message and halts:

```
Null size for device partition sd.si(,,1)
```

If any of the above messages are displayed, the user should verify that the DEFAULTSWAP variable has been correctly set up and that the signal and power cableing on the target disk system is connected and intact.

Once passed the above checks, the test fills a 128 Kbyte block of physical memory with a repeating pattern of 0xb6d9, writes the data buffer to disk block 0 then reads the disk data back into another 128 Kbyte memory buffer.

If the call to the write function returns an error, the test will print the following message and halt:

```
Write Error:  device sd.si(,,1), fd = 3, count = 0x0
target block  = 0
block address = 0x00000000
transfer size = 0x00020000
write buffer  = 0xff0c0000 - 0xff0dffff
```

After the disk write has been performed, the test zeroes out the buffer that will be used during the read, then reads the disk into the buffer.

```
Read Error:  device sd.si(,,1), fd = 3, count = 0x0
target block  = 0
block address = 0x00000000
transfer size = 0x00020000
read buffer   = 0xfffe0000 - 0xffffffff
```

After the disk read has been completed, the test verifies that the write buffer data matches the read buffer data. If there is a data miscompare, the test will print the following message and halt:

```
Data Compare Error:  device = sd.si(,,1), fd = 3
target block  = 0
block address = 0x00000000
write buffer  = 0xff0c0000 - 0xff0dffff
read buffer   = 0xfffe0000 - 0xffffffff
error occurred at buffer offset 0x100
  expected: (0xff0c0100) = 0xb6
  actual:   (0xfffe0100) = 0x00
```

3.18 Tape Write/Read Test (must be prompted)

The Tape write/read test verifies the ability of the Solbourne workstation to write data to a tape and read it back. The test is device independent in that it will run on any tape device supported by the standalone driver.

This test must be prompted in order to run. When the test is not prompted, the following message is printed:

```
Test 17 skipped - use PROMPT command to enable
```

If the test is prompted, the user is prompted for the following information:

```
Enter tape device name:
```

The user must enter the name of the device to test (assuming the user enters **st.si(4,)**). If the user enters "quit" the test will be halted, otherwise the test will attempt to open the specified device. If the open fails, the following message will be printed and the user will be prompted to specify another device name.

```
Device st.si(,4,) cannot be opened
```

If the device is successfully opened, the user is asked to insert a scratch tape and enter return:

```
Please install a scratch tape and hit return when ready to continue
```

When the user enters return, the test writes random data patterns on the tape, reads them back and verifies that the data is correct.

When the disk test begins, the following message is printed:

```
Testing st.si(,4,):
```

The test then begins the following operations:

```
Open the device for write
Write 128 Kbytes of random data patterns
Close the device
Open the device for read
Read the data back
Close the device
Verify the data
```

During the test, one of the following error messages could be displayed depending on the type of error (write, read, compare):

```
Device st.si(,4,) cannot be opened for write

Write Error:  device st.si(,4,), fd = 3, count = 0x0
target block  = 36504
block address = 0x011d3000
transfer size = 0x00003686
write buffer  = 0xffc04840 - 0xffc07ec8

Device 'devicename' cannot be opened for read

Read Error:   device st.si(,4,), fd = 3, count = 0x0
target block  = 36504
block address = 0x011d3000
transfer size = 0x00003686
read buffer   = 0xffa3a6fe - 0xffa3dd86

Data Compare Error:   device st.si(,4,), fd = 3
target block  = 36504
block address = 0x011d3000
transfer size = 0x00003686
write buffer  = 0xffc04840 - 0xffc07ec8
read buffer   = 0xffa3a6fe - 0xffa3dd86
error occurred at buffer offset 0x0
  expected: (0xffc07ec8) = 0xd9
  actual:   (0xffa3dd86) = 0x00
```

Section 4: Commands

4.1 Introduction

This section offers printed copies of man pages for all commands associated with **rdg**(1). The commands are presented in the UNIX man page reference format.

A summary of the command usage is displayed on-line when **rdg** is running by typing:

```
RDG> ?
```

The following is a listing of the **rdg** commands available in this section:

SECTION 1: COMMANDS

between(1)
config(1)
continue(1)
errlim(1)
errors(1)
fbconfig(1)
help(1)
limit(1)
loop(1)
menu(1)
names(1)
next(1)
passes(1)
passlim(1)
prompt(1)
quiet(1)
quit(1)
rdg(1)
restart(1)
run(1)
status(1)
tests(1)
time(1)
vmeconf(1)

NAME

between - Set or display between count

SYNOPSIS

between [*count*]

DESCRIPTION

between sets or displays the current setting of the between count. **between** suppresses printing test completed messages to the screen until *count* passes have completed.

When the **status** (1) reset command is used, the **between** *count* is reset to 1.

OPTION

count Specifies the number of test passes that must be completed before a completion message is displayed. By default the **between** *count* is always set to 1.

EXAMPLE

User input in the example is shown in **boldface** type.

The following example illustrates how to set and redisplay the **between** *count*.

```
RDG> between 4
RDG> between
Between count = 4
RDG>
```

SEE ALSO

rdg (1), **passlim** (1), **status** (1)

NAME

config - Display memory configuration file

SYNOPSIS

config

DESCRIPTION

config displays the memory configuration. When **rdg** is invoked it creates a memory configuration table based on the memory configuration information saved in the diagnostic RAM during the power-up self-tests.

EXAMPLES

User input in the examples is shown in **boldface** type.

In the following example, **config** is entered at the RDG prompt. The current contents of the configuration table are displayed.

```
RDG> config
Memory Configuration:

    5 board(s) totaling 80 Mbytes
    Slot 1        16 Mbytes        Base address = 00000000
    Slot 2        16 Mbytes        Base address = 01000000
    Slot 3        16 Mbytes        Base address = 02000000
    Slot 4        16 Mbytes        Base address = 03000000
    Slot 5        16 Mbytes        Base address = 04000000

RDG>
```

SEE ALSO

rdg (1)

NAME

continue - Set or display continue on error flag

SYNOPSIS

continue [*on* | *off*]

DESCRIPTION

continue sets or displays the continue-on-error flag. If no parameters are specified, **continue** displays the current setting of the continue-on-error flag.

The **continue** flag commands tests to continue executing after a test failure occurs. Tests are designed to check the continue flag to determine if test execution should be halted (the default condition) or if the next test case should be executed.

OPTIONS

on Turns on the continue-on-error flag.

off Turns off the continue-on-error flag.

EXAMPLES

User input in the examples is shown in **boldface** type.

The following example causes the current error message enable flag to be displayed.

```
RDG> continue
continue = off
RDG>
```

The following example illustrates how the **continue** flag is changed and redisplayed.

```
RDG> continue on
RDG> continue
continue = on
RDG>
```

SEE ALSO

rdg (1), **status** (1)

NAME

errlim - Set or display error limit

SYNOPSIS

errlim [*limit*]

DESCRIPTION

errlim sets or displays the current setting of the test error *limit*.

OPTION

limit Specifies the number of test errors that can occur before test execution is halted. By default, the *limit* is set to zero (no error limit). However, the error limit may be changed by specifying a new limit value. The limit value must be entered in unsigned decimal format and be between 0 and 2,147,483,647, inclusive.

EXAMPLES

User input in the examples is shown in **boldface** type.

The following example illustrates how to display the current error limit.

```
RDG> errlim
Error limit = 0
RDG>
```

The following example illustrates how to change and re-display the error limit.

```
RDG> errlim 100
RDG> errlim
Error limit = 100
RDG>
```

SEE ALSO

rdg (1), **errors** (1), **status** (1)

NAME

errors - Display error count

SYNOPSIS

errors

DESCRIPTION

errors displays the number of test errors that have occurred since the last **run**(1) command.

EXAMPLE

User input in the example is shown in **boldface** type.

The following example illustrates how to display the error count.

```
RDG> errors
Total test errors = 0
RDG>
```

SEE ALSO

errlim (1), **rdg** (1), **status** (1)

NAME

fbconfig - displays the frame buffer configuration file

SYNOPSIS

fbconfig

DESCRIPTION

The frame buffer configuration is read from the diagnostic RAM when **rdg** is invoked.

The boards must be configured in descending slot order.

EXAMPLE

User input in the example is shown in **boldface** type.

```
RDG> fbconfig

Frame Buffer Configuration:

    1 graphics board(s):
    Slot    IO address    Board Type    Resolution
    1        81000000     monochrome       low
Slot number of default board to test: 1
RDG>
```

SEE ALSO

rdg(1)

NAME

help - Display this command list or information on a specific command

SYNOPSIS

help [*command* . . .]

DESCRIPTION

The **help** command with no arguments causes a list of command and command usages to be displayed. This is equivalent to the **?** command.

The **help** command with an argument causes the command usage for the specified command to be displayed.

OPTIONS

command

name of command for which help is desired.

EXAMPLE

The following example causes the command usage for the **tests** command to be displayed:

```
RDG> help tests
      Usage: tests [ all | test ... | test:test ... ]
RDG>
```

SEE ALSO

rdg(1)

NAME

limit - Display or set memory test limits

SYNOPSIS

limit [reset | *memname* [low high | reset]]

DESCRIPTION

limit displays or sets the memory test limits of the system. By default, **limit** displays all the memory limits.

limit is set to the amount of installed memory for each memory devices in the system. Memory devices include physical memory, VMEbus address map memory, and VMEbus resident memory boards.

The test programs examine the memory limits to determine how much memory to test.

OPTION

reset Resets the limits back to the default settings. The default settings are determined by the amount of installed memory. For example, as set by the **config** (1) command.

low high

low is the first address and **high** is the last address to test, inclusive.

EXAMPLE

User input in the example is shown in **boldface** type.

The following example displays the current limit settings for all the memory devices.

```
RDG> limit
Memory limits:    LOW        HIGH
     mem     =     5ffff       8ffff
     vmemap  =       40         7ff
     vmemem  =        3         5ff
```

The following example resets the memory limits to their default values.

```
RDG> limit reset
RDG> limit
Memory limits:    LOW        HIGH
     mem     =    40000     1ffffff
     vmemap  =       20         7ff
     vmemem  =        0       7ffff
```

The following example sets the memory limits for physical memory to the range 40000 through 2ffffff hex and sets the VMEbus address map limits to 20 through ff hex, inclusive.

```
RDG> limit mem 40000 2ffffff
RDG> limit vmemap 20 ff
RDG> limit
Memory limits:    LOW       HIGH
     mem     =    40000     2fffff
     vmemap  =      20        ff
     vmemem  =       0      7ffff
```

The following example resets only the VMEbus address map limits to their default values. The physical memory values are not modified.

```
RDG> limit vmemap reset
RDG> limit
Memory limits:    LOW       HIGH
     mem     =    40000     2fffff
     vmemap  =      20        7ff
     vmemem  =       3        5ff

RDG>
```

SEE ALSO

config(1), **rdg**(1)

NAME

loop - Set or display loop on test flag

SYNOPSIS

loop [*on* | *off*]

DESCRIPTION

loop sets or displays the loop on error flag. If no parameters are specified, **loop** displays the current setting of the loop flag.

The loop flag commands tests to loop on the failing test case in the event a test error occurs. Tests are designed to halt when errors occur so that the loop command may be entered.

OPTIONS

on Turns on the loop flag.

off Turns off the loop flag.

EXAMPLES

User input in the examples is shown in **boldface** type.

The following example causes the current loop flag do be displayed.

```
RDG> loop
loop = off
RDG>
```

The following example illustrates how the loop flag is changed and re-displayed.

```
RDG> loop on
RDG> loop
loop = on
RDG>
```

SEE ALSO

rdg(1), **status**(1)

NAME

menu - Display listing of available tests

SYNOPSIS

menu

DESCRIPTION

menu lists the names of all available tests in the default order of execution. **menu** displays tests in the default order of execution.

EXAMPLE

User input in the example is shown in **boldface** type.

The following example displays the list of installed tests.

```
RDG> menu
RDG> menu
Test 1: RTC-58321 real time clock test
Test 2: Memory Data RAM test (affected by prompt)
Test 3: Memory ECC RAM test (affected by prompt)
Test 4: VMEbus address map RAM test
Test 5: VMEbus data path test
Test 6: VMEbus address path test
Test 7: RF3500 SCSI data path (Write Buffer) test
Test 8: I/O ASIC register access test
Test 9: I/O ASIC FIFO/ECC test
Test 10: 7990 LANCE initialization test
Test 11: 7990 LANCE internal loopback test
Test 12: 7990 LANCE external loopback test (must be prompted)
Test 13: 33C93 SBIC (SCSI) enable test
Test 14: 33C93 SBIC (SCSI) data path (Write Buffer) test
Test 15: Ethernet tftp read test
Test 16: Disk write/read test
Test 17: Tape write/read test (must be prompted)
RDG>
```

SEE ALSO

rdg(1), **tests**(1)

NAME

names - Enable or disable printing of test names during test execution

SYNOPSIS

names [*on* | *off*]

DESCRIPTION

names enables or disables the printing of test names during test execution.

OPTIONS

on Enables the printing of the test names during test execution. This is the default setting.

off Disables the printing of the test names during test execution.

EXAMPLES

User input in the examples is shown in **boldface** type.

The following example causes the state of the name flag to be displayed.

```
RDG> names
names = on
RDG>
```

The following example illustrates how the names flag is changed and redisplayed.

```
RDG> names off
RDG> names
names = off
RDG>
```

SEE ALSO

rdg (1), **status** (1)

NAME

next - Execute next selected test

SYNOPSIS

next

DESCRIPTION

next causes the test sequence to be continued, starting with the next selected test. It is used when a test halts on an error and the user wishes to continue test execution with the next test in the sequence.

EXAMPLE

User input in the example is shown in **boldface** type.

In the following example **run** was entered to begin test execution. The current test selection was executed until an error was encountered in test 3. **next** was entered to continue the test sequence starting with the next test in the sequence.

```
RDG> run
Starting Test 1: (testname)
Starting Test 2: (testname)
Starting Test 3: (testname)
Test 3 error: (error message)

RDG> next
Starting Test 4: (testname)
Starting Test 5: (testname)
Starting Test 6: (testname)
      .
      .
      .
Starting Test n: (testname)

Tests completed: Passes = 1 Errors = 1 \
                 Fri Sep 28 15:41:29 1988
RDG>
```

SEE ALSO

between(1), **errlim**(1), **passlim**(1), **rdg**(1), **restart**(1), **run**(1)

NAME

passes - Display pass count

SYNOPSIS

passes

DESCRIPTION

passes displays the number of complete test passes that have made since the last **run** command.

EXAMPLE

User input in the example is shown in **boldface** type.

The following example illustrates how to use the **passes** command.

```
RDG> passes
Total passes = 0
RDG>
```

SEE ALSO

passlim (1), **rdg** (1)

NAME

passlim - Set or display pass limit

SYNOPSIS

passlim [*limit* **]**

DESCRIPTION

passlim sets or displays the current setting of the test pass *limit*. **passlim** specifies the number of test passes that can occur before test execution is halted.

This command should be used when it is desired to execute numerous passes of the test sequence.

OPTION

limit Sets the number of test passes that will be run. By default, *limit* is set to one. *Limit* must be entered in unsigned decimal format in the range 0-to-2,147,483,647, inclusive. A *limit* of 0 specifies that tests execute continuously until a Control-C is entered.

EXAMPLES

User input in the examples is shown in **boldface** type.

The following example illustrates how to display the current pass limit.

```
RDG> passlim
Pass limit = 1
RDG>
```

The following example illustrates how to change and re-display the pass limit.

```
RDG> passlim 0
RDG> passlim
Pass limit = 0
RDG>
```

SEE ALSO

passes (1), **rdg** (1)

NAME

prompt - Set or display prompt flags

SYNOPSIS

prompt [**all** | **off** | *test test* ... | *test:test* ...]

DESCRIPTION

prompt sets or displays the prompt flag for each test program. The command allows the user to selectively alter the default behavior of the test programs by turning the flag for the specified tests on or off.

Only a few of the **rdg** tests use the **prompt** flag. The behavior of the test depends on what the test is attempting to accomplish. In some case, if a test isn't prompted it does not execute. In others, it modifies the test algorithm.

Single tests or a range of tests may be prompted by specifying the test numbers or range of tests number.

The **menu** (1) command indicates which tests examine their **prompt** flags.

OPTIONS

all Set prompt flags for all tests. **all** can be specified at any time to prompt all tests.

off Turns prompt flags for all tests off. **off** can be specified at any time to turn off prompts for all tests.

test Prompt specified *test*. If *test* is not specified, the **prompt** command displays the current status of the **prompt** flags.

EXAMPLES

User input in the examples is shown in **boldface** type.

The following example illustrates how to display the **prompt** flags.

```
RDG> prompt
no prompted tests
RDG> prompt all
RDG> prompt
prompted tests:    1    2    3    4    5    6    7
                   8    9   10   11   12   13   14
                  15   16   17
RDG> prompt off
RDG> prompt 8 9 10
RDG> prompt
prompted tests:   8    9   10
```

SEE ALSO

menu (1), **rdg** (1), **tests** (1)

NAME

quiet - Set or display error message enable flag

SYNOPSIS

quiet [*on* | *off*]

DESCRIPTION

quiet sets or displays the error message enable flag. If no parameters are specified, **quiet** displays the current setting of the flag.

The error message enable flag prevents error messages from being displayed on test failures. This feature should be used to create the tightest possible loop when the **loop** flag is on. A Control-C must be entered to stop the loop and return to the RDG> prompt.

OPTIONS

on Turns on the **quiet** flag.

off Turns off the **quiet** flag.

EXAMPLES

User input in the examples is shown in **boldface** type.

The following example causes the current error message enable flag to be displayed.

```
RDG> quiet
quiet = off
RDG>
```

The following example illustrates how the **quiet** flag is changed and redisplayed.

```
RDG> quiet on
RDG> quiet
quiet = on
RDG>
```

SEE ALSO

rdg (1), **status** (1)

NAME

quit - Exit from **rdg** debugger program

SYNOPSIS

quit

DESCRIPTION

quit exits from the **rdg** debugger program and returns the user to the ROM> prompt.

SEE ALSO

rdg(1)

NAME

rdg - description of the extended ROM resident diagnostics

SYNOPSIS

rdg

DESCRIPTION

rdg is a ROM-resident diagnostics. The test controller provides the commands necessary to randomly select and execute any or all of the available test programs. The operator has control over test execution and can command test programs to loop on error or repeat execution indefinitely.

Command names and abbreviations are not case sensitive.

The acceptable commands follow (bold, uppercase letters represent the abbreviated usage of the command name):

?	Print summary of **rdg** commands
between	Set or display between count
config	Display memory configuration
continue	Set or display continue on error flag
errlim	Set or display error limit
errors	Display error count
fbconfig	Displays the frame buffer configuration
help	Display command list or information on a specific command
limit	Display or set memory test limits
loop	Set or display loop on test flag
menu	Display listing of the available tests
names	Enable or disable printing of test names during test execution
next	Execute next selected test
passes	Display pass count
passlim	Set or display pass limit
prompt	Set or display prompt flags
quiet	Set or display error message enable flag
quit	Exit from **rdg** program
restart	Restart execution of selected tests
run	Start execution of selected tests
status	Display or reset state of modes, flags and counts
tests	Select or display tests to be executed
time	Set a display print time flag
vmeconf	Configure VMEbus devices

NAME

restart - Restart execution of selected tests

SYNOPSIS

restart

DESCRIPTION

restart causes the current **test**(1) selection to be executed beginning with the first test current test selection. The major difference between **restart** and **run**(1) is that **restart** goes back to the first test in the sequence, while **run** continues execution with the next selected test.

The number of times the test selection is executed depends on the value of the **passlim** (1) *limit*.

EXAMPLE

User input in the example is shown in **boldface** type.

In the following example **run** was entered to begin test execution. The current test selection were executed until an error was encountered in test 3. **restart** was entered to start the test sequence again from the beginning.

```
RDG> run
Starting Test 1: (testname)
Starting Test 2: (testname)
Starting Test 3: (testname)
Test 3 error: (error message)

RDG> restart
Starting Test 1: (testname)
Starting Test 2: (testname)
Starting Test 3: (testname)
      .
      .
      .
Starting Test n: (testname)

Tests completed:  Passes = 1    Errors = 0    \
                Fri Sep 28 16:30:33 1988
RDG>
```

SEE ALSO

next(1), **passlim**(1), **rdg**(1), **run**(1)

NAME

run - Start execution of selected tests

SYNOPSIS

run

DESCRIPTION

run causes the current **test**(1) selection to be executed. The number of times the test selection is executed depends on the value of the **passlim** (1) *limit*.

EXAMPLE

User input in the example is shown in **boldface** type.

In the following example **run** was entered to begin test execution. The current test selection was executed once (passlim = 1) followed by a tests completed message. If **passlim**'s *limit* is set to a value other than one, the complete test sequence would be repeatedly executed until *limit* is reached, at which time the program would return to the RDG> prompt. The test completed message is displayed after each pass.

```
RDG> run
Starting Test 1: (testname)
Starting Test 2: (testname)
Starting Test 3: (testname)
      .
      .
      .
Starting Test n: (testname)

Tests completed:  Passes = 1    Errors = 0    \
            Fri Sep 28 16:30:33 1988
RDG>
```

SEE ALSO

next(1), **passlim**(1), **rdg**(1), **restart**(1)

NAME

status - Display or reset state of modes, flags, and counts

SYNOPSIS

status [reset] [flags]

DESCRIPTION

status displays the current state of all modes, program flags, and counters. **flags** resets all the flags, which includes names, continue, loop, and quiet.

OPTION

reset Resets the status of flags, counts, and limits to the default setting. **reset** also resets the test selection back to default values.

flags Resets the status of flags to the default settings.

EXAMPLE

User input in the example is shown in **boldface** type.

```
RDG> status
Wed Nov 25 12:45:20 1987
  Names = on
  Continue = off
  Loop = off
  Quiet = off
  Time = off
  Pass count = 0
  Pass limit = 1
  Between count = 1
  Error count = 0
  Error limit = 0
RDG>
```

SEE ALSO

between(1), **continue**(1), **ecc**(1), **errlim**(1), **errors**(1), **loop**(1), **names**(1), **passes**(1), **passlim**(1), **quiet**(1), **rdg**(1), **time**(1)

NAME

tests - Select or display tests to be executed

SYNOPSIS

tests [all | *test test* ... | *test:test* ... **]**

DESCRIPTION

tests select the tests for execution by the **run**(1) command. By default, all tests are selected for execution when the program is initialized.

Single tests or a range of tests may be selected by specifying the test numbers or range of tests number.

OPTIONS

all Execute all the tests. **all** can be specified at any time to reselect all tests.

test Execute specified *test*. If *test* is not specified, the **tests** command displays the current test selection.

EXAMPLES

User input in the examples is shown in **boldface** type.

The following example illustrates how to display the test selection.

```
RDG> tests
 selected tests:   1    2    3    4    5    6    7    8
                   9
RDG>
```

In the following example, tests 1, 5 and 10 are selected and then displayed.

```
RDG> tests 1 5 9
RDG> tests
selected tests:   1    5    9
RDG>
```

In the following example, tests 8 through 1 are selected and displayed. Note that tests may be selected to run in any order.

```
RDG> tests 8:1
RDG> tests
selected tests:   8    7    6    5    4    3    2    1
RDG>
```

In the following example, all installed tests are selected and displayed.

```
RDG> tests all
RDG> tests
 selected tests:   1    2    3    4    5    6    7    8
                   9
RDG>
```

SEE ALSO

next(1), **rdg**(1), **restart**(1), **run**(1)

NAME

time - Set or display print time flag

SYNOPSIS

time [on | off]

DESCRIPTION

time sets or displays the print-time flag. If no parameters are specified, **time** displays the current setting of the print-time flag and the current time and data. The print-time flag controls whether the current time and date is printed when test names are displayed during test execution. The default state of the print-time flag is off (no time printed). If both the names flag and print-time flag are on, the time and date is printed on the line following the test name during test execution.

OPTIONS

on Turns on the print-time flag.

off Turns off the print-time flag.

EXAMPLES

The following example causes the current print-time flag to be displayed:

```
RDG> time
time = off
Fri Sep 28 14:20:00 1988
RDG>
```

The following example illustrates how the print-time flag is changed and redisplayed.

```
RDG> time on
RDG> time
time = on
Fri Sep 28 14:20:00 1988
RDG>
```

SEE ALSO

names (1), **rdg** (1), **status** (1)

NAME

vmeconf - Configure VMEbus devices

SYNOPSIS

vmeconf

DESCRIPTION

vmeconf generates or displays the VMEbus configuration table.

When **rdg** is invoked, it does not ask the user to generate a VMEbus configuration table. Therefore, if the user wishes to run VMEbus tests, this command must be executed first.

vmeconf prompts for all user input. It accepts no options or arguments at the command line.

Currently, **vmeconf** supports the Ciprico Rimfire, Excelan Ethernet, and Plessy RAM boards.

EXAMPLE

User input in the example is shown in **boldface** type.

The following example shows how **vmeconf** is used to remove an Excelan Ethernet VMEbus board from the configuration, then how the program would be used to put the board back into the configuration table.

```
RDG> vmeconf

VMEbus Configuration consists of four boards
      (0) Ciprico Rimfire 3500 VMEbus-to-SCSI
            Am = 0x2d   Addr = 0x5000     Physaddr = 0x85ff5000
      (1) Excelan Ethernet
            Am = 0x3d   Addr = 0xd00000   Physaddr = 0x87d00000
      (2) Plessey RAM (512K)
            Am = 0x3d   Addr = 0x100000   Physaddr = 0x87100000

Do you wish to change this configuration? (y/n) y
Do you want the default configuration? (y/n) n
Do you want to delete any entries? (y/n) y
Entry number to delete (q to quit)? 1
Entry number to delete (q to quit)? q
Do you want to add any entries? (y/n) n

      (0) Ciprico Rimfire 3500 VMEbus-to-SCSI
            Am = 0x2d   Addr = 0x5000     Physaddr = 0x85ff5000
      (2) Plessey RAM (512K)
            Am = 0x3d   Addr = 0x100000   Physaddr = 0x87100000

Do you wish to change this configuration? (y/n) y
Do you want the default configuration? (y/n) n
Do you want to delete any entries? (y/n) n
Do you want to add any entries? (y/n) y
How many vme boards are to be added? (0-5) 1

Enter information for board 1:
```

```
Valid vme board types are:
      0: none
      1: Ciprico Rimfire 3500 VMEbus-to-SCSI
      2: Excelan Ethernet
      3: Plessey RAM (512K)
  Type of board? 2

Valid address modifiers are:
      9: extended user data access
      d: extended supervisor data access
      39: standard user data access
      3d: standard supervisor data access
      29: short user data access
      2d: short supervisor data access
  Address modifier? 3d
  Address? d00000

VMEbus Configuration consists of 3 boards
      (0) Ciprico Rimfire 3500 VMEbus-to-SCSI
            Am = 0x2d   Addr = 0x5000     Physaddr = 0x85ff5000
      (1) Excelan Ethernet
            Am = 0x3d   Addr = 0xd00000   Physaddr = 0x87d00000
      (2) Plessey RAM (512K)
            Am = 0x3d   Addr = 0x100000   Physaddr = 0x87100000

Do you wish to change this configuration? (y/n) n
RDG>
```

SEE ALSO

rdg(1)

Appendix A: MARCH Algorithm

A.1 Introduction

This appendix explains the way the MARCH pattern algorithm works. The steps follow:

1. Every location of the target RAM is written from low address to the high address with the background pattern. This is the first pattern of the initial complement pair (00).
2. The low address in the RAM is read and verified to contain the background pattern. If the background pattern is correct, the complement pattern is written to the same address. The location is read again and verified to now contain the complement pattern.
3. Step 2 is repeated for all addresses, from low to high. At this point, every location in RAM contains the complement pattern. Once this has completed, the test goes in reverse order from the high address back to the low address. The first read is now verified to contain the complement pattern followed by a write and read of the background pattern.
4. This test advances to the next complement pattern pair and repeats the same steps.

Appendix B: Moving Inversions Test Algorithm

B.1 Introduction

This appendix contains a general description of the Moving Inversions Algorithm (MOVI) commonly used to verify the addressing and data integrity of RAM devices. The strengths of this algorithm are its relatively short execution time, functional testing of memory bits, and dynamic tests of best and worst case access times.

In principle, MOVI inverts the data of each address sequentially, thus creating an access time by the jump from one address to ahother which contains different information. To measure access times, the data at each address is read before and after inversion. This requires three operations on each address: a read, a write, and another read.

The read/write/read operations are performed with both forward and backward (reverse) address sequences, and also with n orders of the address-bit significance (where n is the number of address bits).

A general stepwise description of the MOVI algorithm is presented below:

1. The target memory is filled with a background data pattern.
2. In the forward direction (from low to high addresses) a target location is read and check for the correct background pattern (this is the 1st read of the forward pass).
3. The complement pattern (1's complement of the background pattern) is written to the target location.
4. The target location is read and checked for the correct complement pattern (this is the 2nd read of the forward pass).
5. Steps 2 through 4 are repeated for all addresses until the high address of the memory has been reached. At this point the memory should be filled with the complement pattern (assuming no errors were encountered).
6. In the reverse direction (from high to low address) a target location is read and checked for the correct complement pattern (this is the 1st read of the reverse pass).
7. The background pattern (1's complement of the complement pattern) is written to the target location.
8. The target location is read and checked for the correct background pattern (this is the 2nd read of the reverse pass).
9. Steps 6 through 8 are repeated for all addresses until the low address of memory has been reached. At this point the memory should be filled with the background pattern again (assuming no errors were encountered during the reverse pass).

The steps above implement the first iteration of the MOVI test, where the basic address increment value is 1. Successive iterations use higher address-bit significance up to 2**n where n is the number of involved address bits. This is the same as using a different bit of the address each time as the least significant bit for incrementing through all possible addresses. This has the effect of incrementing through all the addresses by 2's, 4's, 8's, and so on; every address overflow generates and end-around carry, so that all addresses are tested once in each sequence.

The table illustrates the binary address sequences generated by MOVI for and eight location memory:

Forward sequences:

Iteration	0	1	2
	lsb	lsb	lsb
	v	v	v
	000 0	000 0	000 0
	001 1	010 2	100 4
	010 2	100 4	——
	011 3	110 6	001 1
	100 4	——	101 5
	101 5	001 1	——
	110 6	011 3	010 2
	111 7	101 5	110 6
	——	111 7	——
		——	011 3
			111 7
			——

Reverse sequences:

Iteration	0	1	2
	lsb	lsb	lsb
	v	v	v
	111 7	111 7	111 7
	110 6	101 5	011 3
	101 5	011 3	——
	100 4	001 1	110 6
	011 3	——	010 2
	010 2	110 6	——
	001 1	100 4	101 5
	000 0	010 2	001 1
	——	000 0	——
		——	100 4
			000 0
			——

Index