MVME162BUG

162Bug Debugging Package

User's Manual

(MVME162BUG/D2)

Notice

While reasonable efforts have been made to assure the accuracy of this document, Motorola, Inc. assumes no liability resulting from any omissions in this document, or from the use of the information obtained therein. Motorola reserves the right to revise this document and to make changes from time to time in the content hereof without obligation of Motorola to notify any person of such revision or changes.

No part of this material may be reproduced or copied in any tangible medium, or stored in a retrieval system, or transmitted in any form, or by any means, radio, electronic, mechanical, photocopying, recording or facsimile, or otherwise, without the prior written permission of Motorola, Inc.

It is possible that this publication may contain reference to, or information about Motorola products (machines and programs), programming, or services that are not announced in your country. Such references or information must not be construed to mean that Motorola intends to announce such Motorola products, programming, or services in your country.

Restricted Rights Legend

If the documentation contained herein is supplied, directly or indirectly, to the U.S. Government, the following notice shall apply unless otherwise agreed to in writing by Motorola, Inc.

Use, duplication, or disclosure by the Government is subject to restrictions as set forth in subparagraph (c)(1)(ii) of the Rights in Technical Data and Computer Software clause at DFARS 252.227-7013.

Motorola, Inc.
Computer Group
2900 South Diablo Way
Tempe, Arizona 85282

Preface

The *MVME162Bug Debugging Package User's Manual* provides general information and a diagnostic firmware guide for the MVME162Bug (162Bug) Debugging Package.

This edition (/D2) covers 162Bug versions 2.1 and up only; and is usable with all versions of the MVME162 and MVME162LX series of microcomputers.

Use of the debugger, the debugger command set, use of the one-line assembler/ disassembler, and system calls for the Debugging Package are all contained in the *Debugging Package for Motorola 68K CISC CPUs User's Manual (68KBUG1/Dx* and *68KBUG2/Dx)*.

This manual is intended for anyone who wants to design OEM systems, supply additional capability to an existing compatible system, or work in a lab environment for experimental purposes.

A basic knowledge of computers and digital logic is assumed.

Note also that for these 68K CISC-chip based debuggers, data sizes are: byte (8 bits), word (16 bits), and longword (32 bits). In addition, commands that act on words or longwords over a range of addresses may truncate the selected range so as to end on a properly aligned boundary.

To use this manual, you should be familiar with the publications listed in the *Related Documentation* section in Chapter 1 of this manual.

The following conventions are used in this document:

bold

is used for user input that you type just as it appears. Bold is also used for commands, options and arguments to commands, and names of programs, directories, and files.

italic

is used for names of variables to which you assign values. Italic is also used for comments in screen displays and examples.

`courier`

is used for system output (e.g., screen displays, reports), examples, and system prompts.

RETURN or **<CR>**

represents the carriage return key.

CTRL

represents the Control key. Execute control characters by pressing the **CTRL** key and the letter simultaneously, e.g., **CTRL-d**.

WARNING

This equipment generates, uses, and can radiate radio frequency energy and if not installed and used in accordance with the documentation for this product, may cause interference to radio communications. It has been tested and found to comply with the limits for a Class A Computing Device pursuant to Subpart J of Part 15 of FCC rules, which are designed to provide reasonable protection against such interference when operated in a commercial environment. Operation of this equipment in a residential area is likely to cause interference in which case the user, at the user's own expense, will be required to take whatever measures necessary to correct the interference.

Motorola® and the Motorola symbol are registered trademarks of Motorola, Inc.

Delta Series, VMEmodule, and VMEsystem are trademarks of Motorola, Inc.

IndustryPack and IP are trademarks of GreenSpring Computers, Inc.

Timekeeper and Zeropower are trademarks of Thompson Components.

UNIX is a registered trademark in the United States and other countries, licensed exclusively through X/Open Company Limited.

All other products mentioned in this document are trademarks or registered trademarks of their respective holders.

DISCLAIMER OF WARRANTY

Unless otherwise provided by written agreement with Motorola, Inc., the software and the documentation are provided on an "as is" basis and without warranty. This disclaimer of warranty is in lieu of all warranties whether express, implied, or statutory, including implied warranties of merchantability or fitness for any particular purpose.

Printed in the United States of America

July 1994

Contents

Description of 162Bug 1-1
162Bug Implementation 1-6
Detailed Installation and Start-Up 1-6
ROMboot 1-8
Memory Requirements 1-8
Diagnostic Facilities 1-9
Manufacturing Test Process 1-10
Related Documentation 1-11
Manual Terminology 1-12
Scope 2-1
Overview of Diagnostic Firmware 2-1
  System Startup 2-1
  Design Features 2-1
Diagnostic Monitor 2-2
  Monitor Start-Up 2-2
  Command Entry and Directories 2-3
Utilities 2-4
  Append Error Messages Mode - Command AEM 2-5
  Clear Error Messages - Command CEM 2-5
  Test Group Configuration (cf) Parameters Editor - Command CF 2-5
  Display Error Counters - Command DE 2-5
  Display Error Messages - Command DEM 2-6
  Display Pass Count - Command DP 2-6
  Help - Command HE 2-6
  Help Extended - Command HEX 2-6
  Loop Always Mode - Prefix LA 2-8
  Loop-Continue Mode - Prefix LC 2-8
  Loop-On-Error Mode - Prefix LE 2-8
  Line Feed Suppression Mode - Prefix LF 2-8
  Loop Non-Verbose Mode - Prefix LN 2-9
  Display/Revise Self Test Mask - Command MASK 2-9
  Non-Verbose Mode - Prefix NV 2-9
  Switch Directories - Command SD 2-10
  Stop-On-Error Mode - Prefix SE 2-10
  Self Test - Command ST 2-10
  Clear (Zero) Error Counters - Command ZE 2-10

Zero Pass Count - Command ZP 2-11

Local RAM (RAM) and Static RAM (SRAM) Tests 3-2

Memory Addressing - ADR 3-3

Alternating Ones/Zeros - ALTS 3-5

Bit Toggle - BTOG 3-6

Code Execution/Copy - CODE 3-8

Data Patterns - PATS 3-9

Local Parity Memory Error Detection - PED 3-11

Permutations - PERM 3-13

Quick Write/Read - QUIK 3-14

Memory Refresh Testing - REF 3-15

Random Data - RNDM 3-17

MK48T0x (RTC) Tests 3-18

BBRAM Addressing - ADR 3-19

Clock Function - CLK 3-21

Battery Backed-Up SRAM - RAM 3-23

Memory Controller Chip (MCC) Tests 3-24

Device Access - ACCESSA 3-26

Register Access - ACCESSB 3-27

Prescaler Clock Adjust - ADJ 3-28

FAST Bit - FAST 3-29

MPU Clock Speed - MPUCS 3-30

Prescaler Clock - PCLK 3-31

DRAM Refresh Timing - DRAM 3-32

Timer Counters - TMR*n*A 3-33

Timer Free-Run - TMR*n*B 3-35

Timer Clear on Compare - TMR*n*C 3-37

Timer Overflow Counter - TMR*n*D 3-38

Timer Interrupts - TMR*n*E 3-40

Vector Base Register - VBR 3-42

Watchdog Timer Counter - WDTMRA 3-43

Watchdog Timer Board Fail - WDTMRB 3-44

Watchdog Timer Local Reset - WDTMRC 3-46

Additional MCC Test Group Messages 3-47

ECC Memory Board (MCECC) Tests 3-48

Check-Bit DRAM - CBIT 3-50

Exceptions - EXCPTN 3-52

Multi-Bit-Error - MBE 3-53

Single-Bit-Error - SBE 3-54

Scrubbing - SCRUB 3-55

MC68040 Internal Cache (DCAC) Tests 3-57

Data Cache Copyback - DCAC_CB 3-58
Data Cache Registers - DCAC_RD 3-60
Data Cache Writethrough - DCAC_WT 3-61
Memory Management Unit (MMU) Tests 3-63
Display Table Search - DISPSRCH 3-65
Build Default Tables - TBLBLD 3-66
Verify Default Tables - TBLVERF 3-67
TC Register Test - TC 3-68
RP Register Test - RP 3-69
Tablewalk Mapped Pages - WALK 3-70
Mapped ROM Read Test - MAPROM 3-71
Used Page Test - USEDPAGE 3-72
Modified Page Test - MODPAGE 3-73
Invalid Page Test - INVPAGE 3-74
Write Protect Page Test - WPPAGE 3-75
VME Interface Chip (VME2) Tests 3-76
Register Access - REGA 3-77
Register Walking Bit - REGB 3-79
Software Interrupts (Polled Mode) - SWIA 3-81
Software Interrupts (Processor Interrupt Mode) - SWIB 3-83
Software Interrupts Priority - SWIC 3-85
Timer Accuracy Test - TACU 3-87
Tick Timer Increment - TMRA, TMRB 3-89
Prescaler Clock Adjust - TMRC 3-90
Tick Timer No Clear On Compare - TMRD, TMRE 3-92
Tick Timer Clear On Compare - TMRF, TMRG 3-94
Overflow Counter - TMRH, TMRI 3-96
Watchdog Timer Counter - TMRJ 3-98
Watchdog Timer Board Fail - TMRK 3-99
LAN Coprocessor for Ethernet (LANC) Tests 3-100
Chip Self Test - CST 3-102
Diagnose Internal Hardware - DIAG 3-104
Dump Configuration/Registers - DUMP 3-106
External Loopback Cable - ELBC 3-107
External Loopback Transceiver - ELBT 3-111
Internal Loopback - ILB 3-114
Interrupt Request - IRQ 3-117
Monitor (Incoming Frames) Mode - MON 3-119
Time Domain Reflectometry - TDR 3-120
Additional Error Messages 3-122
NCR 53C710 SCSI I/O Processor (NCR) Tests 3-126

Device Access - ACC1 3-127
Register Access - ACC2 3-129
DMA FIFO - DFIFO 3-131
Interrupts - IRQ 3-132
Loopback - LPBK 3-134
SCRIPTs Processor test - SCRIPTS 3-135
SCSI FIFO test - SFIFO 3-138
IndustryPack Interface Controller (IPIC) Tests 3-139
Read Internal Registers - ACCESSA 3-140
Write to Internal Registers - ACCESSB 3-141
Interrupt Control Registers - INRPT 3-142
IPIC Error Messages 3-144
Serial Communication Controller (SCC) (Z85230) Tests 3-146
SCC Device/Register Access - ACCESS 3-147
SCC Interrupt Request - IRQ 3-148
SCC Baud Rates - BAUDS 3-149
SCC External Loopback - ELPBCK 3-150
SCC Internal Loopback - ILPBCK 3-151
SCC Modem Control - MDMC 3-152
SCC Error Messages 3-153
Introduction 4-1
Configuring the Board Information Block (CNFG) 4-1
Setting Operational Parameters (ENV) 4-2
Configuring MVME162Bug Parameters 4-2
Configuring the VMEbus Interface 4-9
Configuring the IndustryPacks 4-14

List of Figures

Flow Diagram of 162Bug Board Operational Mode (Sheet 1 of 3) 1-2
Flow Diagram of 162Bug System Operational Mode 1-5
Help Screen 2-7

List of Tables

Diagnostic Utilities 2-4
Diagnostic Test Groups 3-1
RAM and SRAM Test Group 3-2
RTC Test Group 3-18
MCC Test Group 3-24
MCECC Test Group 3-48
DCAC Test Group 3-57
MMU Test Group 3-63
VME2 Test Group 3-76
LANC Test Group 3-100
NCR Test Group 3-126
IPIC Test Group 3-139
IPIC Error Messages 3-144
SCC Test Group 3-146
SCC Error Messages 3-153

162BUG GENERAL INFORMATION 1

This member of the M68000 Firmware family is implemented on the MVME162 or MVME162LX MC68040 or MC68LC040-based embedded controller, and is known as the MVME162BUG, or 162Bug. It includes diagnostics for testing and configuring IndustryPack modules.

Description of 162Bug

162Bug consists of three parts:

- ❑ A command-driven, user-interactive software debugger, described in the *Debugging Package for Motorola 68K CISC CPUs User's Manual* and hereafter referred to as "the debugger" or "162Bug"
- ❑ A command-driven diagnostic package for the MVME162 hardware, described in Chapters 2 and 3, and hereafter referred to as "the diagnostics"
- ❑ A user interface that accepts commands from the system console terminal

When using 162Bug, you operate out of either the debugger directory or the diagnostic directory. If you are in the debugger directory, the debugger prompt `162-Bug>` displays and you have all of the debugger commands at your disposal. If you are in the diagnostic directory, the diagnostic prompt `162-Diag>` displays and you have all of the diagnostic commands at your disposal as well as all of the debugger commands. You can switch between directories with the Switch Directories (**SD**) command, or you can examine the commands in the current directory with the Help (**HE**) command.

Because 162Bug is command-driven, it performs its operations in response to commands you enter at the keyboard. The flow of control in 162Bug is shown in Figures 1-1 and 1-2. When you enter a command, 162Bug executes the command and the prompt reappears. However, if you enter a command that causes execution of user target code (for example, **GO**), then control may or may not return to 162Bug, depending on the outcome of the user program.

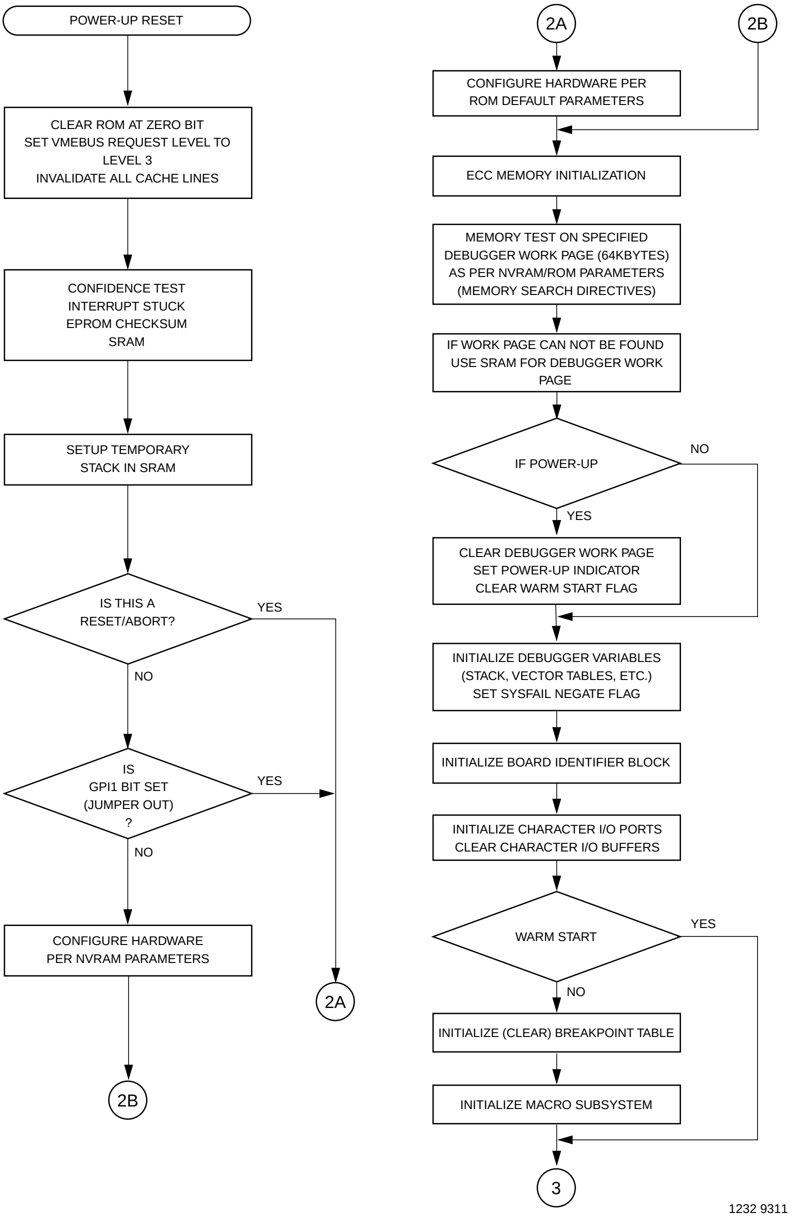
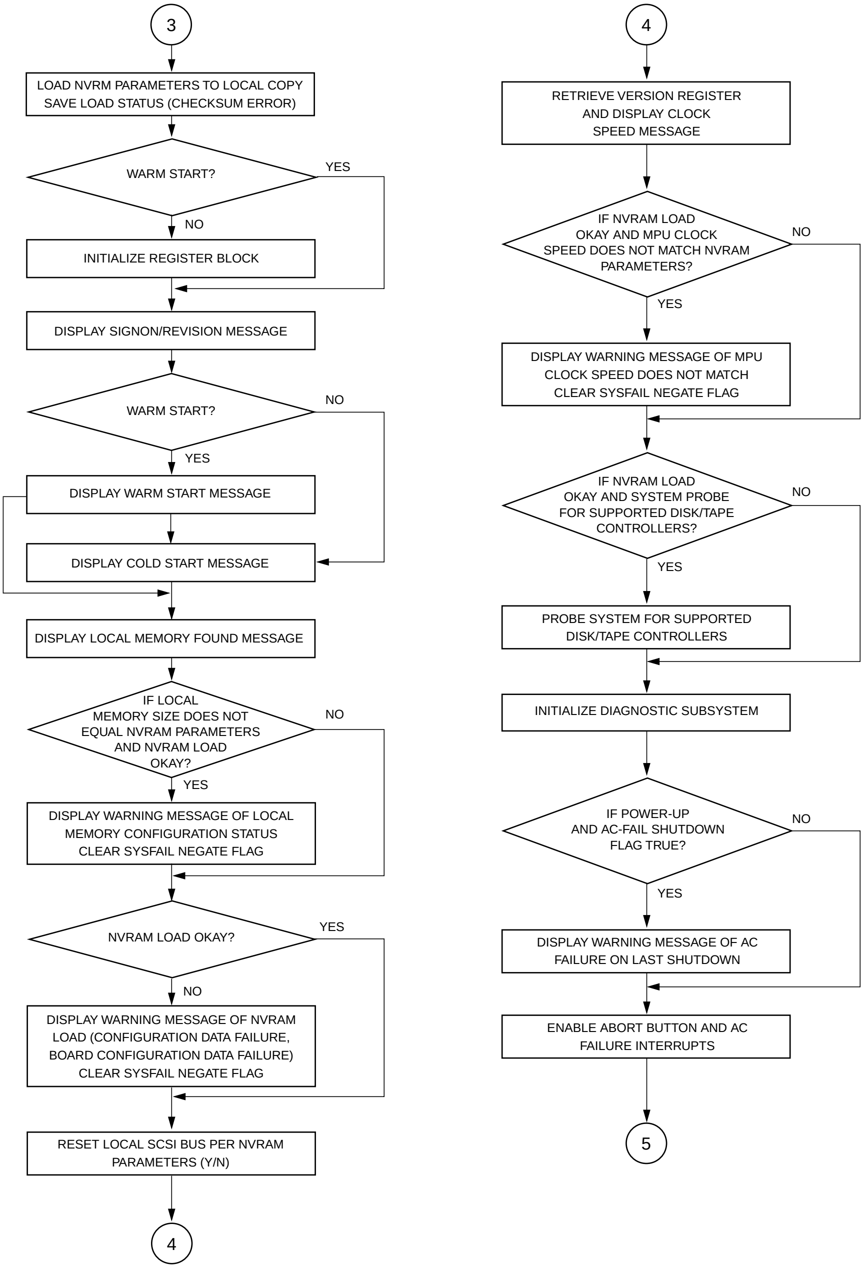

POWER-UP RESET
CLEAR ROM AT ZERO BIT
SET VMEBUS REQUEST LEVEL TO LEVEL 3
INVALIDATE ALL CACHE LINES
CONFIDENCE TEST
INTERRUPT STUCK
EPROM CHECKSUM
SRAM
SETUP TEMPORARY STACK IN SRAM
IS THIS A RESET/ABORT?
YES
NO
IS GPI1 BIT SET (JUMPER OUT) ?
YES
NO
CONFIGURE HARDWARE PER NVRAM PARAMETERS
2A
2B
2A
2B
CONFIGURE HARDWARE PER ROM DEFAULT PARAMETERS
ECC MEMORY INITIALIZATION
MEMORY TEST ON SPECIFIED DEBUGGER WORK PAGE (64KBYTES) AS PER NVRAM/ROM PARAMETERS (MEMORY SEARCH DIRECTIVES)
IF WORK PAGE CAN NOT BE FOUND USE SRAM FOR DEBUGGER WORK PAGE
IF POWER-UP
NO
YES
CLEAR DEBUGGER WORK PAGE
SET POWER-UP INDICATOR
CLEAR WARM START FLAG
INITIALIZE DEBUGGER VARIABLES (STACK, VECTOR TABLES, ETC.)
SET SYSFAIL NEGATE FLAG
INITIALIZE BOARD IDENTIFIER BLOCK
INITIALIZE CHARACTER I/O PORTS
CLEAR CHARACTER I/O BUFFERS
WARM START
YES
NO
INITIALIZE (CLEAR) BREAKPOINT TABLE
INITIALIZE MACRO SUBSYSTEM
3
1232 9311

Figure 1-1. Flow Diagram of 162Bug Board Operational Mode (Sheet 1 of 3)


3
LOAD NVRM PARAMETERS TO LOCAL COPY
SAVE LOAD STATUS (CHECKSUM ERROR)
WARM START?
YES
NO
INITIALIZE REGISTER BLOCK
DISPLAY SIGNON/REVISION MESSAGE
WARM START?
NO
YES
DISPLAY WARM START MESSAGE
DISPLAY COLD START MESSAGE
DISPLAY LOCAL MEMORY FOUND MESSAGE
IF LOCAL MEMORY SIZE DOES NOT EQUAL NVRAM PARAMETERS AND NVRAM LOAD OKAY?
NO
YES
DISPLAY WARNING MESSAGE OF LOCAL MEMORY CONFIGURATION STATUS CLEAR SYSFAIL NEGATE FLAG
NVRAM LOAD OKAY?
YES
NO
DISPLAY WARNING MESSAGE OF NVRAM LOAD (CONFIGURATION DATA FAILURE, BOARD CONFIGURATION DATA FAILURE) CLEAR SYSFAIL NEGATE FLAG
RESET LOCAL SCSI BUS PER NVRAM PARAMETERS (Y/N)
4
4
RETRIEVE VERSION REGISTER AND DISPLAY CLOCK SPEED MESSAGE
IF NVRAM LOAD OKAY AND MPU CLOCK SPEED DOES NOT MATCH NVRAM PARAMETERS?
NO
YES
DISPLAY WARNING MESSAGE OF MPU CLOCK SPEED DOES NOT MATCH CLEAR SYSFAIL NEGATE FLAG
IF NVRAM LOAD OKAY AND SYSTEM PROBE FOR SUPPORTED DISK/TAPE CONTROLLERS?
NO
YES
PROBE SYSTEM FOR SUPPORTED DISK/TAPE CONTROLLERS
INITIALIZE DIAGNOSTIC SUBSYSTEM
IF POWER-UP AND AC-FAIL SHUTDOWN FLAG TRUE?
NO
YES
DISPLAY WARNING MESSAGE OF AC FAILURE ON LAST SHUTDOWN
ENABLE ABORT BUTTON AND AC FAILURE INTERRUPTS
5

fc015 9212

Figure 1-1. Flow Diagram of 162Bug Board Operational Mode (Sheet 2 of 3)

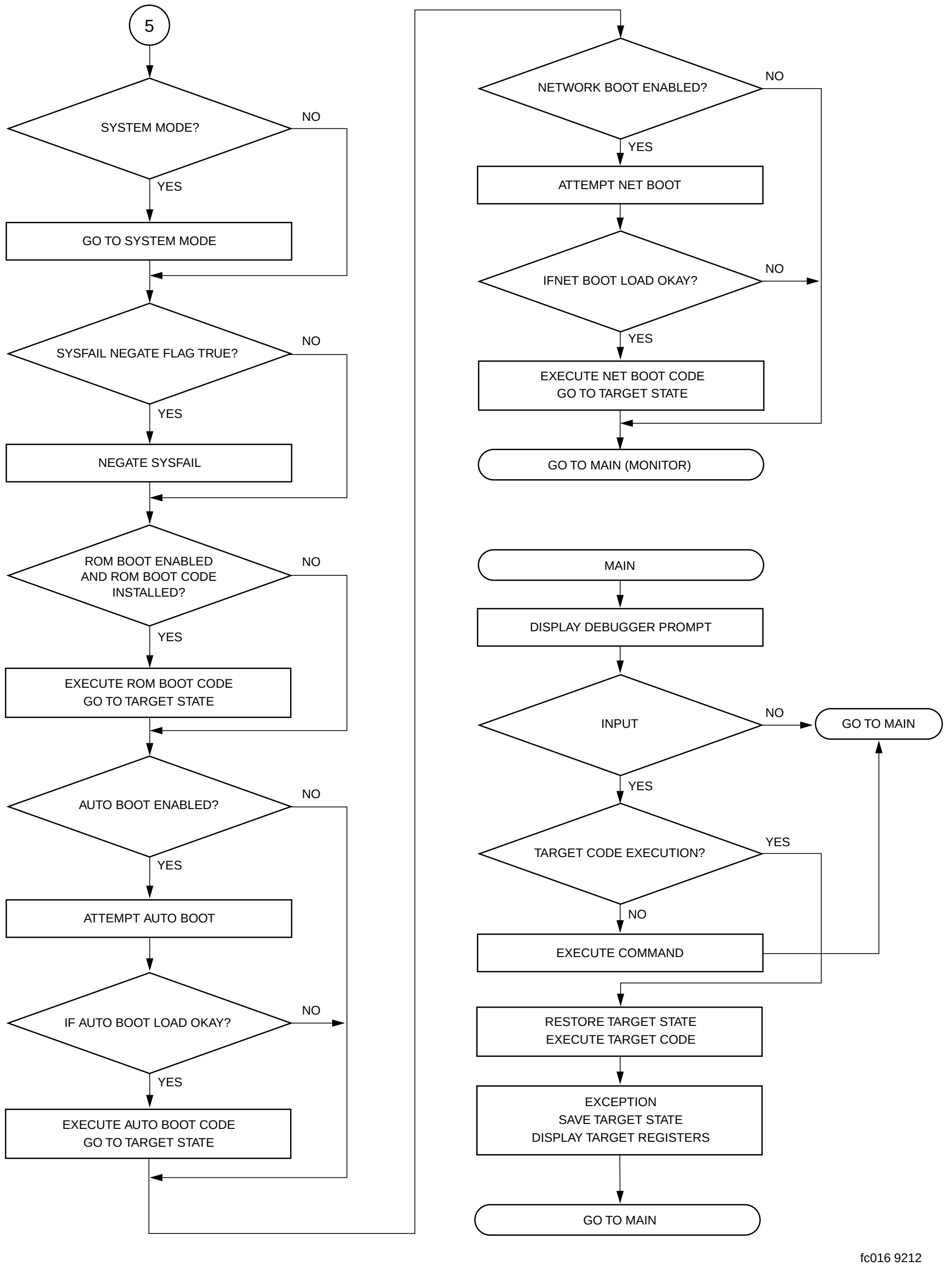


Figure 1-1. Flow Diagram of 162Bug Board Operational Mode (Sheet 3 of 3)

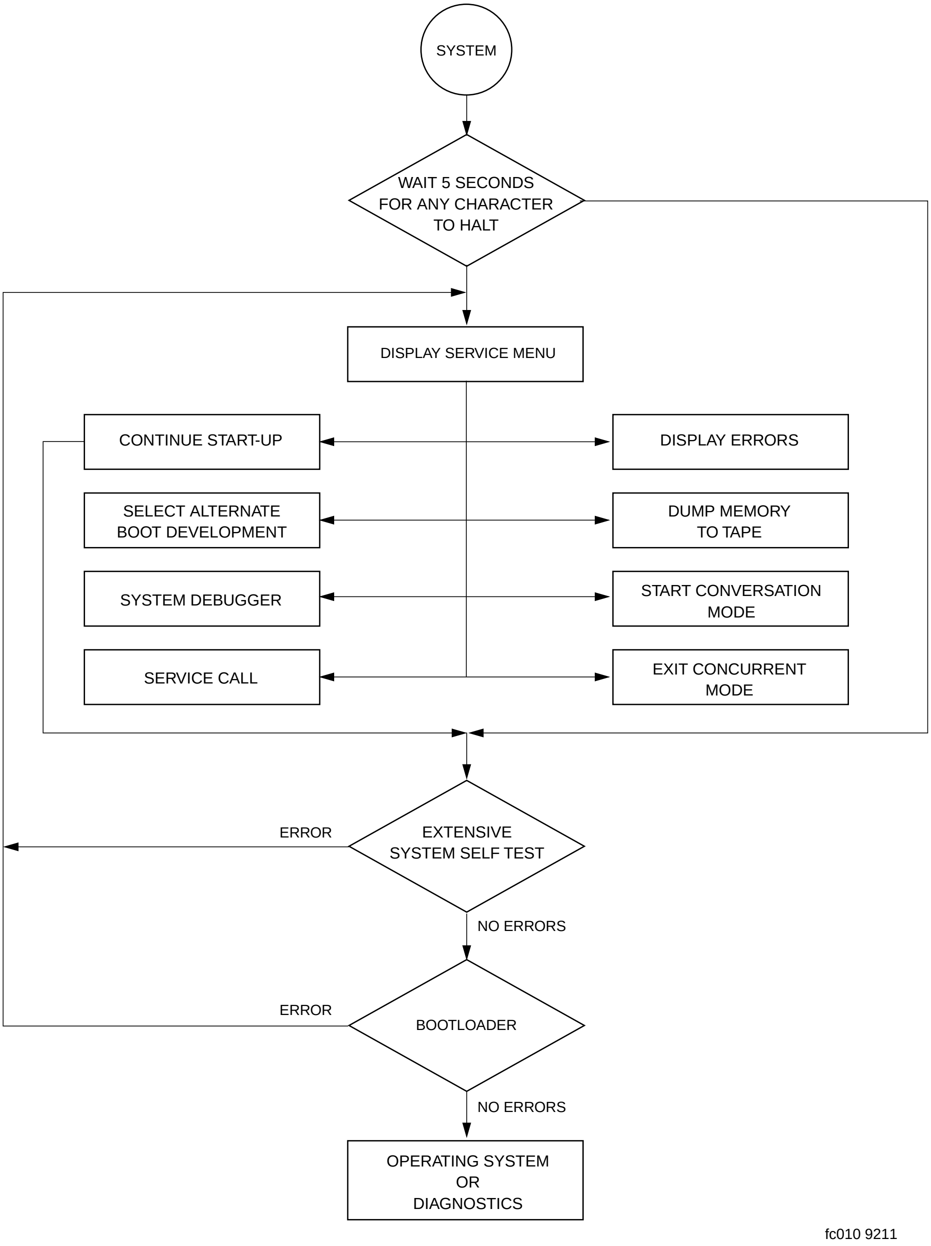


Figure 1-2. Flow Diagram of 162Bug System Operational Mode

162Bug Implementation

Physically, on MVME162-0*xx* series modules, 162Bug is contained in two of the four 28F020 FLASH memories, providing 512KB (128K longwords) of storage. Optionally, the 162Bug can be loaded and executed in a single 27C040 PLCC PROM.

On the MVME162LX (MVME162-2*xx*) series modules, 162Bug is contained in a single 27C040 DIP EPROM installed in socket XU24.

Detailed Installation and Start-Up

Even though 162Bug is installed on the MVME162 module, for 162Bug to operate properly with the MVME162, you must follow the general setup procedure described in the *Debugging Package for Motorola 68K CISC CPUs User's Manual*, *AND* the steps below:

Caution **Inserting or removing modules while power is applied could damage module components.**

1. Turn all equipment power OFF. Refer to the *MVME162* (or *MVME162LX) Embedded Controller User's Manual* and install/remove jumpers on headers as required for your particular application.

   Jumpers on header J22 (on MVME162 series) or J11 (MVME162LX series) affect 162Bug operation as listed below. The default condition for the MVME162-0XX is with all eight jumpers installed, between pins 1-2, 3-4, 5-6, 7-8, 9-10, 11-12, 13-14, and 15-16. For the MVME162-2xx (MVME162LX), the default is with seven jumpers installed, but no jumper on J11, pins 7-8. These readable jumpers can be read as a register (at $FFF4202D) on the Memory Controller Chip (MCchip) ASIC. The bit values are read as a one when the jumper is off, and as a zero when the jumper is on. Refer also to the *MVME162* (or *MVME162LX) Embedded Controller Programmer's Reference Guide* for more information on the MCC.

   The MVME162BUG reserves/defines the four lower order bits (GPI3 to GPI0) as follows:

NOTE: The bits are swapped (header pins reversed) between the 162 and the 162LX.

Bit	J22 Pins (on 162-0xx)	J11 Pins (on 162-2xx)	Description
Bit #0 (GPI0)	15-16	1-2	When this bit is a one (high), it instructs the debugger to use local Static RAM for its work page, i.e., variables, stack, vector tables, etc.
Bit #1 (GPI1)	13-14	3-4	When this bit is a one (high), it instructs the debugger to use the default setup/operation parameters in ROM versus the user setup/operation parameters in Non-Volatile RAM (NVRAM). This is the same as depressing the RESET and ABORT switches at the same time. This feature can be used in the event the user setup is corrupted or does not meet a sanity check. Refer to the **ENV** command for the ROM defaults.
Bit #2 (GPI2)	11-12	5-6	Reserved for future use.
Bit #3 (GPI3)	9-10	7-8	When this bit is a zero (low), it informs the debugger that it is executing out of the FLASH memories. When this bit is a one (high), it informs the debugger that it is executing out of the PROM.
Bit #4 (GPI4)	7-8	9-10	Open to your application.
Bit #5 (GPI5)	5-6	11-12	Open to your application.
Bit #6 (GPI6)	3-4	13-14	Open to your application.
Bit #7 (GPI7)	1-2	15-16	Open to your application.

2. Configure header J1 by installing/removing a jumper between pins 1 and 2. A jumper installed/removed enables/disables the system controller function of the MVME162.

3. (Does NOT apply to MVME162LX series) You may configure Port B of the Z85230 serial communications controller via a serial interface module (SIM) which is installed at connector J10 on the MVME162 board. Four serial interface modules are available:

   - EIA-232-D DTE (SIM05)
   - EIA-232-D DCE (SIM06)
   - EIA-530 DTE (SIM07)
   - EIA-530 DCE (SIM08)

   For information on removing and/or installing a SIM, refer to the *MVME162 Embedded Controller User's Manual*.

4. (Does NOT apply to MVME162LX series) Jumpers on headers J11 and J12 configure serial ports 1 and 2 to drive or receive clock signals provided by the TXC and RXC signal lines. The factory configures the module for asynchronous communication, that is, installs no jumpers. Refer to the *MVME162 Embedded Controller User's Manual* if your application requires configuring ports 1 and 2 for synchronous communication.

5. (Does NOT apply to MVME162LX series) If using a PROM version of the 162Bug, install the PROM device in socket U47. Be sure that the physical chip orientation is correct, that is, with the flatted corner of the PROM aligned with the corresponding portion of the PROM socket on the MVME162 module.

   Check the jumper installation on header J21 for correct size. Connect pins 1 and 2 on J21 for 27C080 devices, or pins 2 and 3 for 27C040 devices. The factory default is 2 and 3.

   Remove the jumper on J22 pins 9 and 10.
6. Connect the terminal that is to be used as the 162Bug system console to the default debug EIA-232-D port at serial port 1 on the front panel of the MVME162 module. Refer to the *MVME162* (or *MVME162LX) Embedded Controller User's Manual* for other connection options.
7. (Applies ONLY to MVME162LX series) The EPROM/Flash header J12 must be set to configuration 3, with jumpers between J12 pins 5 and 6, 8 and 10, and 9 and 11. This sets it up for 512K x 8 EPROMs.

ROMboot

On the MVME162 series modules, as shipped from the factory, 162Bug occupies the first half of the FLASH memory. This leaves the second half of the FLASH memory and the PROM socket (U47) available for your use. The 162Bug is also available in PROM if your application requires all of the FLASH memory. Contact your Motorola sales office for assistance.

On the MVME162LX series modules, 162Bug occupies an EPROM installed in socket XU24, leaving three sockets available (XU21 - XU23) and the FLASH.

Memory Requirements

The program portion of 162Bug is approximately 512KB of code, consisting of download, debugger, and diagnostic packages and contained entirely in FLASH or PROM.

The 162Bug executes from $FF800000 whether in FLASH or PROM. With jumper at J22 pins 9-10 installed (factory ship configuration for MVME162-0xx), the FLASH memories appear at address FF800000 and are the parts executed during reset. With this configuration, the PROM socket is mapped to address $FFA00000. If you remove the jumper at J22 pin 9 and 10, the address spaces of the FLASH and PROM are swapped. For the MVME166-2xx (MVME162LX), factory ship is with jumper J11 pins 7-8 removed (162Bug operates out of EPROM).

The 162Bug initial stack completely changes 8KB of SRAM memory at addresses offset $C000 from the SRAM base address, at power-up or reset.

Type of Memory Present	Default DRAM Base Address	Default SRAM Base Address
A single DRAM mezzanine	$00000000	$FFE00000 (onboard SRAM)
A single SRAM mezzanine	N/A	$00000000
A DRAM mezzanine stacked with an SRAM mezzanine	$00000000	$E1000000
Two DRAM mezzanines stacked	$00000000	$FFE00000 (onboard SRAM)

DRAM can be ECC or parity type. DRAM mezzanines are mapped in contiguously starting at zero ($00000000), largest first. With two mezzanines of the same size, ECC type DRAM is first. If both are ECC type, the bottom one is first.

The 162Bug requires 2KB of NVRAM for storage of board configuration, communication, and booting parameters. This storage area begins at $FFFC16F8 and ends at $FFFC1EF7.

162Bug requires a minimum of 64KB of contiguous read/write memory to operate. The **ENV** command controls where this block of memory is located. Regardless of where the onboard RAM is located, the first 64KB is used for 162Bug stack and static variable space and the rest is reserved as user space. Whenever the MVME162 is reset, the target PC is initialized to the address corresponding to the beginning of the user space, and the target stack pointers are initialized to addresses within the user space, with the target Interrupt Stack Pointer (ISP) set to the top of the user space.

Diagnostic Facilities

The 162Bug package includes a set of hardware diagnostics for testing and troubleshooting the MVME162. To use the diagnostics, switch directories to the diagnostic directory. If you are in the debugger directory, you can switch to the diagnostic directory with the debugger command Switch Directories (**SD**). The diagnostic prompt (`162-Diag>`) appears. Refer to Chapter 2 for complete descriptions of the diagnostic routines available and instructions on how to invoke them. Note that some diagnostics depend on restart defaults that are set up only in a particular restart mode. The documentation for such diagnostics includes restart information.

Manufacturing Test Process

During the manufacturing process for MVME162 modules, the manufacturing test parameters and testing state flags are stored in NVRAM. These strings are installed during the manufacturing process and result in the product performing manufacturing tests. None of these tests harm the product or system into which a module is installed. Entering an ASCII break on the console port from a terminal terminates these tests.

The two state flags that start the test processes are:

```
FLASH EMPTY$00122984
```

and

```
Burnin test$00000000
```

If either string is in the first location of NVRAM ($FFFC0000), the test process starts.

This note is to inform users about the manufacturing test process. It is not intended to instruct customers in its use; Motorola reserves the right to delete, change, or modify this process.

Related Documentation

The following publications are applicable to 162Bug and may provide additional information. If they are not shipped with this product, you may purchase them through your Motorola sales office. Obtain non-Motorola documents from the sources listed.

Document Title	Motorola Publication Number
M68040 Microprocessor User's Manual	M68040UM
MVME162 Embedded Controller User's Manual	MVME162
MVME162LX Embedded Controller User's Manual	MVME162LX
MVME162 Embedded Controller Programmer's Reference Guide	MVME162PG
MVME162LX Embedded Controller Programmer's Reference Guide	MVME162LXPG
MVME162 Embedded Controller Support Information	SIMVME162
MVME162LX Embedded Controller Support Information	SIMVME162LX
Single Board Computers SCSI Software User's Manual	SBCSCSI
Debugging Package for Motorola 68K CISC CPUs User's Manual	68KBUG1 and 68KBUG2
MVME712M Transition Module and MVME147P2 Adapter Board User's Manual	MVME712M
MVME712A/MVME712AM/MVME712B Transition Module and MVME147P2 Adapter Board User's Manual	MVME712A

Note **Although not shown in the preceding list, each Motorola Computer Group manual publication number is suffixed with characters that represent the revision level of the document, such as "/D2" (the second revision of a manual); a supplement bears the same number as a manual but has a suffix such as "/D2A1" (the first supplement to the second revision of the manual).**

The following publications are available from the sources indicated.

ANSI Small Computer System Interface-2 (SCSI-2), Draft Document X3.131-198X, Revision 10c; Global Engineering Documents, P.O. Box 19539, Irvine, CA 92714.

Versatile Backplane Bus: VMEbus, ANSI/IEEE Std 1014-1987, The Institute of Electrical and Electronics Engineers, Inc., 345 East 47th Street, New York, NY 10017 (VMEbus Specification). This is also available as *Microprocessor system bus for 1 to 4 byte data, IEC 821 BUS*, Bureau Central de la Commission Electrotechnique Internationale; 3, rue de Varembé, Geneva, Switzerland.

Manual Terminology

Throughout this manual, data and address parameters are preceded by a character that specifies the numeric format as follows:

$	dollar	specifies a hexadecimal character
%	percent	specifies a binary number
&	ampersand	specifies a decimal number

Unless otherwise specified, all address references are in hexadecimal throughout this manual.

An asterisk (*) following names of signals that are *level significant* denotes that the signal is *true* or valid when the signal is low.

An asterisk (*) following the names of signals that are *edge significant* denotes that the actions initiated by that signal occur on high to low transition.

In this manual, *assertion* and *negation* are used to specify forcing a signal to a particular state. In particular, *assertion* and *assert* refer to a signal that is active or *true*; *negation* and *negate* indicate a signal that is inactive or *false*. These terms are used independently of the voltage level (high or low) that they represent.

Data and address sizes are defined as follows:

❏ A *byte* is eight bits, numbered 0 through 7, with bit 0 being the least significant.

❏ A *word* is 16 bits, numbered 0 through 15, with bit 0 being the least significant.

❏ A *longword* is 32 bits, numbered 0 through 31, with bit 0 being the least significant.

DIAGNOSTIC FIRMWARE 2

Scope

This chapter contains information about the operation and use of the MVME162 Diagnostic Firmware Package, hereinafter referred to as "the diagnostics". The *Diagnostic Monitor* section in this chapter gives you guidance in setting up the system and invoking the utilities and tests. The *Utilities* section describes the utilities.

The diagnostic tests themselves are described in Chapter 3.

Overview of Diagnostic Firmware

The MVME162 diagnostic firmware package is contained in the same programmed devices as the 162Bug. The diagnostics package is a complete diagnostic monitor, comprising a battery of utilities and tests for exercise, test, and debug of hardware in the MVME162 environment. The diagnostics are menu driven and include a Help (**HE**) command, which displays a menu of all available diagnostic functions, that is, the tests and utilities. Several tests have a subtest menu that you can call with the **HE** command. In addition, some utilities have subfunctions, and, as such, have subfunction menus.

System Startup

Refer to the *Detailed Installation and Startup* section in Chapter 1.

Design Features

Design features of the diagnostic firmware are as follows:

Assembly Language Low-level hardware interface code is written in assembly language to control the hardware exactly. Where possible, the C programming language is used to improve readability and portability.

Bug Interface The diagnostic package shares ROM space with the 162Bug, but the interface between these programs is minimal and well defined.

Compatibility The user interface to the MVME162 diagnostic package is similar to existing diagnostic packages. If you are familiar with a package, you should be able to use this test set without study.

Menu Driven The user interface is menu driven. A Help (**HE**) command is available for each test or set of tests in the package.

Diagnostic Monitor

The tests described in this manual are called, commands are input, and results reported by means of a common diagnostic monitor (the system monitor used for 162Bug), hereafter called *monitor*. This monitor is command line driven and provides input/output facilities, command parsing, error reporting, interrupt handling, and a multi-level directory for menu selection.

Monitor Start-Up

When the monitor is first brought up, either by power up or pressing the RESET switch, it displays the following on the diagnostic video display terminal (port 1 terminal):

```
Copyright Motorola Inc. 1992, All Rights Reserved

MVME162 Debugger/Diagnostics Release Version x.x - mm/dd/yy
COLD Start

Local Memory Found =00400000 (&4194304)

MPU Clock Speed =25Mhz

162-Bug>
```

If, after a delay, the 162Bug begins to display test result messages on the bottom line of the screen in rapid succession, the MVME162 is in the Bug system mode. If this is not the desired mode of operation, then press the ABORT switch. When the menu is displayed, enter a **3** to go to the system debugger. You can change the environment with the Set Environment to Bug/Operating System (**ENV**) command. Refer to the *Debugging Package for Motorola 68K CISC CPUs User's Manual* for details of Bug operation in the system mode.

At the `162-Bug>` prompt, enter **SD** to switch to the diagnostics directory. The prompt should now read `162-Diag>`. The Switch Directories (**SD**) command is described elsewhere in this chapter.

Command Entry and Directories

Enter commands when the prompt `162-Diag>` appears. Enter the *mnemonic* name for the command, then press the carriage return key **<CR>**. You can enter multiple commands on a single command line. If a command expects parameters and another command is to follow it, separate the two with a semicolon (**;**). For example, to invoke the command **RAM ADR** and the **RTC CLK** command on a single command line, enter **RAM ADR ; RTC CLK**. Spaces are not required before or after the semicolon but are shown here for legibility. Spaces are required between commands and their arguments.

Several commands consist of a command name that is listed in a main (root) directory and a subcommand that is listed in the sub directory for that particular command. In the main directory are commands such as **RAM** and **VME2**. These commands are used to refer to a set of lower level commands. To call up a particular **RAM** test, enter (on the same line) **RAM ADR**. This command causes the monitor to find the **RAM** subdirectory, and then to execute the command (test) **ADR** from that subdirectory.

Examples:

Root-Level Commands:

HE	Help
DE	Display Error Counters

Subdirectory-Level Commands:

RAM ADR	Random Access Memory Tests (directory), Memory Addressing test
VME2 REGB	VMEchip2 Tests (directory), Register Walking Bit test

The **RAM** and **VME2** directories in these examples are test group names. If the first part of a command is a test group name, you can enter any number and/or sequence of tests from that test group after the test group name so long as the bug's input buffer size limit is not exceeded. For example, to execute the **ADR** and **PATS** tests from the **RAM** directory, enter **RAM ADR PATS**.

Utilities

In addition to individual or sets of tests, the diagnostic package provides the utilities (root-level commands or general commands) listed in the next table and described on the following pages.

Table 2-1. Diagnostic Utilities

Mnemonic	Description
AEM	Append Error Messages Mode
CEM	Clear Error Messages
CF	Test Group Configuration (cf) Parameters Editor
DE	Display Error Counters
DEM	Display Error Messages
DP	Display Pass Count
HE	Help
HEX	Help Extended
LA	Loop Always Mode
LC	Loop-Continue Mode
LE	Loop-On-Error Mode
LF	Line Feed Suppression Mode
LN	Loop Non-Verbose Mode
MASK	Display/Revise Self Test Mask
NV	Non-Verbose Mode
SD	Switch Directories
SE	Stop-On-Error Mode
ST	Self Test
ZE	Clear (Zero) Error Counters
ZP	Zero Pass Count

Append Error Messages Mode - Command AEM

This command allows you to accumulate error messages in the internal error message buffer of the diagnostic monitor. The **AEM** command sets the internal append error messages flag of the diagnostic monitor. When the internal append error messages flag is clear, the diagnostic error message buffer is erased (cleared of all character data) before each test is executed. The duration of this command is for the life of the command line being parsed by the diagnostic monitor. The default of the internal append error messages flag is clear. The internal flag is not set until it is encountered in the command line by the diagnostic monitor.

Clear Error Messages - Command CEM

This command allows you to manually clear the internal error message buffer of the diagnostic monitor.

Test Group Configuration (cf) Parameters Editor - Command CF

The **cf** parameters control the operation of all tests in a test group. For example, the **RAM** test group has parameters such as starting address, ending address, parity enable, etc. At the time of initial execution of the diagnostic monitor, the default configuration parameters are copied from the firmware into the debugger work page. Here you can modify the configuration parameters with the **CF** command.

When you invoke the **CF** command, you are prompted with a brief parameter description and the current value of the parameter. You may enter a new value for that parameter, or a carriage return to proceed to the next configuration parameter. You may specify one or more test groups as argument(s) immediately following the **CF** command on the command line. If no arguments follow the **CF** command, the parameters for all test groups are presented.

Display Error Counters - Command DE

Each test or command in the diagnostic monitor has an individual error counter. As errors are encountered in a particular test, that error counter is incremented. If you were to run a self-test or a series of tests, the results could be broken down as to which tests passed by examining the error counters. To display all errors, enter **DE**. **DE** displays the results of a particular test if the name of that test follows **DE**. Only nonzero values are displayed.

Display Error Messages - Command DEM

This command allows you to manually display (dump) the internal error message buffer of the diagnostic monitor.

Display Pass Count - Command DP

A count of the number of passes in Loop-Continue (**LC**) mode is kept by the monitor. This count is displayed with other information at the conclusion of each pass. To display this information without using **LC**, enter **DP**.

Help - Command HE

On-line documentation has been provided in the form of a Help command (syntax: **HE [*command name*]**). This command displays a menu of the top level directory of utility commands and test group names if no parameters are entered, or the menu of a subdirectory if the name of that subdirectory is entered. (The top level directory lists "`(DIR)`" after the name of each command that has a subdirectory.) For example, to bring up a menu of all the memory tests, enter **HE RAM**. When a menu is too long to fit on the screen, it pauses until you press the carriage return, **<CR>**, again. To review a description of an individual test, enter the full name. For example, **HE RAM CODE** displays information on the RAM Code Execution/Copy test routine. The Help screen is shown in Figure 2-1.

Help Extended - Command HEX

The **HEX** command provides an interactive, continuous mode of the **HE** command. The syntax is **HEX<CR>**. The prompt displayed for **HEX** is the question mark (`?`). You may then type the name of a directory or command. Type **QUIT** to exit.

```
162-Diag>he
AEM      Append Error Messages Mode
CEM      Clear Error Messages
CF       Configuration Editor
DCAC     MC68040 Data Cache Tests (DIR)
DE       Display Errors
DEM      Display Error Messages
DP       Display Pass Count
HE       Help on Tests/Commands
HEX      Help Extended
IPIC     IP Interface Controller (IPIC ASIC) Tests (DIR)
LA       Loop Always Mode
LANC     LAN Coprocessor (Intel 82596) Tests (DIR)
LC       Loop Continuous Mode
LE       Loop on Error Mode
LF       Line Feed Mode
LN       Loop Non-Verbose Mode
MASK     Self Test Mask
MCC      Memory Controller Chip (MCC ASIC) Tests (DIR)
MCECC    ECC Memory Board Diagnostics (DIR)
MMU      MC68040 MMU Tests (DIR)
NCR      NCR 53C710 SCSI I/O Processor Test (DIR)
NV       Non-Verbose Mode
Press "RETURN" to continue
```

```
RAM      Random Access Memory Tests (DIR)
RTC      MK48T0x Timekeeping (DIR)
SCC      Serial Communication Controller (Z85230)Tests(DIR)
SE       Stop on Error Mode
SRAM     Static Random Access Memory Tests (DIR)
ST       Self Test (DIR)
VME2     VME2Chip2 Tests (DIR)
ZE       Zero Errors
ZP       Zero Pass Count
162-Diag>
```

Figure 2-1. Help Screen

Loop Always Mode - Prefix LA

To endlessly repeat a failed test, enter the prefix **LA**. The **LA** command has no effect until a test failure occurs. To break the loop, press the BREAK key on the diagnostic video display terminal. Certain tests disable the BREAK key interrupt, so pressing the ABORT or RESET switches on the MVME162 front panel may be necessary.

Loop-Continue Mode - Prefix LC

To endlessly repeat a test or series of tests, enter the prefix **LC**. This loop includes everything on the command line. To break the loop, press the BREAK key on the diagnostic video display terminal. Certain tests disable the BREAK key interrupt, so pressing the ABORT or RESET switches on the MVME162 front panel may be necessary.

Loop-On-Error Mode - Prefix LE

If you are using an oscilloscope or logic analyzer, you may want to endlessly repeat a test (loop) while an error is detected. If you include the **LE** command on the command line, a failed test is re-executed as long as the previous execution returned a failure status. To break the loop, press the BREAK key on the diagnostic video display terminal. Certain tests disable the BREAK key interrupt, so pressing the ABORT or RESET switches on the MVME162 front panel may be necessary.

Line Feed Suppression Mode - Prefix LF

The **LF** command sets the internal line feed mode flag of the diagnostic monitor. The default state of the internal line feed mode flag is clear, which causes the executing test title/status line(s) to be terminated with a line feed character (scrolled). The duration of the **LF** command is the life of the user command line in which it appears. The line feed mode flag is normally used by the diagnostic monitor when executing a system mode selftest. Although rarely invoked as a user command, the **LF** command is available to the diagnostic user.

Loop Non-Verbose Mode - Prefix LN

The **LN** command modifies the way that a failed test is endlessly repeated. The **LN** command has no effect until a test failure occurs. Including **LN** in the command line suppresses further printing of the test title and pass/fail status. This is useful for more rapid execution of the failing test; that is, the **LN** command contributes to a "tighter" loop.

Display/Revise Self Test Mask - Command MASK

The syntax is:

MASK [*TEST NAME*]

where *TEST NAME* is the name of a diagnostic test.

MASK is used with an argument to enable/disable a test from running under Self Test. If **mask** is invoked with NO arguments, the currently disabled tests are displayed.

When the **mask** command is used on an MVME162 system, the mask values are preserved in non-volatile memory. This allows the system to be completely powered down without disturbing the Self Test mask.

If the **mask** command is invoked with a parameter, the parameter must be a specific test name, for example, **mask ram adr**.

The **mask** command is a "toggle" command - if the specified test name mask was SET, it will be RESET; if it was RESET, it will be SET. After the toggle, the new Self Test mask is displayed.

If the **mask** command is invoked with an invalid test name or a test directory (as opposed to a specific test name), it outputs an appropriate error message.

The **mask** command may be invoked with NO parameters, in which case it displays the current Self Test mask.

Non-Verbose Mode - Prefix NV

Upon detecting an error, the tests display a substantial amount of data. To suppress the scrolling display, invoke Non-Verbose mode, which suppresses all messages except `PASSED` or `FAILED`. Invoke the Non-Verbose mode by entering **NV** before a command name. For example, **NV ST** runs the self-test, but only shows the names of the subtests and the results (pass/fail).

Switch Directories - Command SD

To leave the diagnostic directory (and disable the diagnostic tests), enter **SD**. At this point, only the commands for 162Bug function. When in the 162Bug directory, the prompt reads `162-Bug>`. To return to the diagnostic directory, enter the command **SD** again. When in the diagnostic directory, the prompt reads `162-Diag>`. The purpose of this feature is to allow you to access 162Bug without the diagnostics being visible.

Stop-On-Error Mode - Prefix SE

It is sometimes desirable to stop a test or series of tests at the point where an error is detected. **SE** accomplishes that for most of the tests. To invoke the command, enter **SE** before the test or series of tests that is to run in Stop-On-Error mode.

Self Test - Command ST

The monitor provides an automated test mechanism called self test. This mechanism runs all the tests included in an internal self-test directory. The command **HE ST** lists the top level of the self test directory in alphabetical order. It lists each test for a particular command in the section pertaining to the command.

When in system mode, use the **HE ST** command to execute the suite of tests that are run at system mode start up. This command is useful for debugging board failures that may require toggling between the test suite and Bug. When the test suite completes, the Bug prompt is displayed, ready for other commands. For details on extended confidence test operation, refer to the *Debugging Package for Motorola 68K CISC CPUs User's Manual*.

Clear (Zero) Error Counters - Command ZE

The error counters originally come up with the value of zero, but it is occasionally desirable to reset them to zero at a later time. This command resets all of the error counters to zero. The error counters can be individually reset by entering the specific test name following the command. Example: **ZE VME2 TMRA** clears the error counter associated with **VME2 TMRA**.

Zero Pass Count - Command ZP

Invoking the **ZP** command resets the pass counter to zero. This is frequently desirable before typing in a command that invokes the Loop-Continue mode. Entering this command on the same line as **LC** results in the pass counter being reset every pass.

TEST DESCRIPTIONS 3

Detailed descriptions of 162Bug's diagnostic tests are presented in this chapter. The test sets are described in the order shown in the following table.

Table 3-1. Diagnostic Test Groups

Test Set	Description
RAM	Local RAM Tests
SRAM	Static RAM Tests
RTC	MK48T0x Real-Time Clock Tests
MCC	Memory Controller Chip Tests
MCECC	ECC Memory Board (MCECC) Tests
DCAC	MC68040 Internal Data Cache Tests
MMU	Memory Management Unit Tests
VME2	VME Interface Chip Tests
LANC	LAN Coprocessor (Intel 82596) Tests
NCR	NCR 53C710 SCSI I/O Processor Tests
IPIC	IndustryPack Interface Chip Tests
SCC	Serial Communication Controller (Z85230) Tests

Local RAM (RAM) and Static RAM (SRAM) Tests

These sections describe the individual **RAM** and **SRAM** tests. The **SRAM** tests are identical in function to the corresponding tests in the **RAM** test group but are executed over the range of Static RAM on the MVME162.

Entering **RAM** or **SRAM** without parameters causes all **RAM** or **SRAM** tests to execute in the order shown in the table below, except as noted.

To run an individual test, add that test name to the **RAM** or **SRAM** command. The individual tests are described in alphabetical order on the following pages.

Table 3-2. RAM and SRAM Test Group

Mnemonic	Description
QUIK	Quick Write/Read
ALTS	Alternating Ones/Zeros
PATS	Data Patterns
ADR	Memory Addressing
CODE	Code Execution/Copy
PERM	Permutations
RNDM	Random Data
BTOG	Bit Toggle
Bypassed during SRAM testing:	
PED	Parity Error Detection
REF	Memory Refresh

Memory Addressing - ADR

Verifies addressing of memory in the range specified by the configuration parameters for the **RAM** test group. Addressing errors are sought by using a memory locations address as the data for that location. This test is coded to use only 32-bit data entities. The test proceeds as follows:

1. A Locations Address is written to its location (*n*).
2. The next location (*n*+4) is written with its address complemented.
3. The next location (*n*+8) is written with the most significant (MS) 16 bits and least significant (LS) 16 bits of its address swapped with each other.
4. Steps 1, 2, and 3 are repeated throughout the specified memory range.
5. The memory is read and verified for the correct data pattern(s) and any errors are reported.
6. The test is repeated using the same algorithm as above (steps 1 through 5) except that inverted data is used to insure that every data bit is written and verified at both "0" and "1".

Command Input:

```
162-Diag>RAM ADR
```

or:

```
162-Diag>SRAM ADR
```

Response/Messages:

Note that in all responses shown below, the response "RAM " is RAM or SRAM, depending upon which test set is being performed. After the command has been issued, the following line is printed:

```
RAM     ADR: Addressability......................... Running --->
```

If all parts of the test are completed correctly, then the test passes.

```
RAM     ADR: Addressability......................... Running ---> PASSED
```

If the test fails, then the display appears as follows:

```
RAM     ADR: Addressability......................... Running ---> FAILED

Data Miscompare Error:
Address =________, Expected =________, Actual =________
```

Alternating Ones/Zeros - ALTS

Verifies addressing of memory in the range specified by the configuration parameters for the **RAM** test group. Addressing errors are sought by using the address of a memory location as the data for that location. This test is coded to use only 32-bit data entities. The test proceeds as follows:

1. Location (*n*) is written with data of all bits 0.
2. The next location (*n*+4) is written with all bits 1.
3. Steps 1 and 2 are repeated throughout the specified memory range.
4. The memory is read and verified for the correct data pattern(s) and any errors are reported.

Command Input:

`162-Diag>`**RAM ALTS**

or:

`162-Diag>`**SRAM ALTS**

Response/Messages:

Note that in all responses shown below, the response "`RAM`" is `RAM` or `SRAM`, depending upon which test set is being performed.

After the command has been issued, the following line is printed:

```
RAM     ALTS: Alternating Ones/Zeroes............... Running --->
```

If all parts of the test are completed correctly, then the test passes.

```
RAM     ALTS: Alternating Ones/Zeroes............... Running ---> PASSED
```

If the test fails, then the display appears as follows:

```
RAM     ALTS: Alternating Ones/Zeroes............... Running ---> FAILED
```

```
Data Miscompare Error:
Address =________, Expected =________, Actual =________
```

Bit Toggle - BTOG

Verifies addressing of memory in the range specified by the RAM test directory configuration parameters. (Refer to the **CF** command.) The RAM test directory configuration parameters also determine the value of the global random data seed used by this test.

The global random data seed is incremented after it is used by this test. This test uses the following test data pattern generation algorithm:

1. Random data seed is copied into a work register.
2. Work register data is shifted right one bit position.
3. Random data seed is added to work register using unsigned arithmetic.
4. Data in the work register may or may not be complemented.
5. Data in the work register is written to current memory location.

If the RAM test directory configuration parameter for code cache enable equals "Y", the microprocessor code cache is enabled. This test is coded to operate using the 32-bit data size only. Each memory location in the specified memory range is written with the test data pattern. Each memory location in the specified memory range is then written with the test data pattern complemented before it is written. The memory under test is read back to verify that the complement test data is properly retained. Each memory location in the specified memory range is then written with the test data pattern. The memory under test is read back to verify that the test data is properly retained.

Command Input:

```
162-Diag>RAM BTOG
```

or:

```
162-Diag>SRAM BTOG
```

Response/Messages:

Note that in all responses shown below, the response "RAM " is RAM or SRAM, depending upon which test set is being performed.

After the command has been issued, the following line is printed:

```
RAM     BTOG: Bit Toggle........................... Running --->
```

If all parts of the test are completed correctly, then the test passes.

```
RAM     BTOG: Bit Toggle........................... Running ---> PASSED
```

If the test fails, then the display appears as follows:

```
RAM     BTOG: Bit Toggle............................ Running ---> FAILED

Data Miscompare Error:
Address =________, Expected =________, Actual =________
```

Code Execution/Copy - CODE

Verifies memory by copying test code to memory and executing. The code in the memory under test copies itself to the next higher memory address and executes the new copy. This process is repeated until there is not enough memory, as specified by the configuration parameters, to perform another code copy and execution.

Command Input:

```
162-Diag>RAM CODE
```

or:

```
162-Diag>SRAM CODE
```

Response/Messages:

Note that in all responses shown below, the response "RAM " is RAM or SRAM, depending upon which test set is being performed.

After the command has been issued, the following line is printed:

```
RAM     CODE: Code Execution/Copy................... Running --->
```

If all parts of the test are completed correctly, then the test passes.

```
RAM     CODE: Code Execution/Copy................... Running ---> PASSED
```

The test failure mode is typified by the nondisplay of the PASSED message above after more than about 1 minute, which indicates that the MPU has irrecoverably crashed. Hardware reset is required to recover from this error.

Data Patterns - PATS

Verifies memory by writing a pattern to an address, reading it back, and verifying it. If the test address range (test range) is less than 8 bytes, the test immediately returns pass status. The effective test range end address is reduced to the next lower 8-byte boundary if necessary. Memory in the test range is filled with all ones ($FFFFFFFF). For each location in the test range, the following patterns are used:

$00000000
$01010101
$03030303
$07070707
$0F0F0F0F
$1F1F1F1F
$3F3F3F3F
$7F7F7F7F

Each location in the test range is, individually, written with the current pattern and the 1's complement of the current pattern. Each write is read back and verified. This test is coded to use only 32-bit data entities.

Command Input:

162-Diag>**RAM PATS**

or:

162-Diag>**SRAM PATS**

Response/Messages:

Note that in all responses shown below, the response "RAM " is RAM or SRAM, depending upon which test set is being performed.

After the command has been issued, the following line is printed:

```
RAM     PATS: Patterns............................. Running --->
```

If all parts of the test are completed correctly, then the test passes.

```
RAM     PATS: Patterns............................. Running ---> PASSED
```

If the test fails, then the display appears as follows:

```
RAM     PATS: Patterns.............................. Running ---> FAILED

Data Miscompare Error:
Address =________, Expected =________, Actual =________
```

Local Parity Memory Error Detection - PED

Verifies specified memory locations with parity interrupt disabled and enabled. The memory range and address increment is specified by the RAM test directory configuration parameters. (Refer to the **CF** command.)

First, each memory location has the data portion verified by writing/verifying all zeros, and all ones. Each memory location is tested once with parity interrupt disabled, and once with parity interrupt enabled. Parity checking is enabled, and data is written and verified at the test location that causes the parity bit to toggle on and off (verifying that the parity bit of memory is good). Next, data with incorrect parity is written to the test location. The data is read, and if a parity error exception does occur, the fault address is compared to the test address. If the addresses are the same, the test passed and the test location is incremented until the end of the test range has been reached.

Command Input:

`162-Diag>`**RAM PED**

or:

`162-Diag>`**SRAM PED**

Response/Messages:

Note that in all responses shown below, the response "`RAM` " is `RAM` or `SRAM`, depending upon which test set is being performed.

After the command has been issued, the following line is printed:

```
RAM       PED:  Local Parity Memory Detection.... Running --->
```

If all parts of the test are completed correctly, then the test passes.

```
RAM       PED:  Local Parity Memory Detection.... Running ---> PASSED
```

If any part of the test fails, then the display appears as follows:

```
RAM       PED:  Local Parity Memory Detection.... Running ---> FAILED
(error message)
```

Here, (error message) is one of the following:

If a data verification error occurs:

```
RAM/PED Test Failure Data:

Data Miscompare Error:
Address =________, Expected =________, Actual =________
```

If an unexpected exception, such as a parity error being detected as the parity bit was being toggled:

```
RAM/PED Test Failure Data:

Unexpected Exception Error, Vector =________
Address Under Test =________
```

If no exception occurred when data with bad parity was read:

```
RAM/PED Test Failure Data:

Parity Error Detection Exception Did Not Occur

Exception Vector =________
Address Under Test =________
```

If the exception address was different from that of the test location:

```
RAM/PED Test Failure Data:

Fault Address Miscompare, Expected =________, Actual =________
```

Permutations - PERM

Verifies that the memory in the test range can accommodate 8-, 16-, and 32-bit writes and reads in any combination. The test range is the memory range specified by the **RAM** test group configuration parameters for starting and ending address. If the test address range (test range) is less than 16 bytes, the test immediately returns pass status. The effective test range end address is reduced to the next lower 16-byte boundary if necessary.

This test performs three data size test phases in the following order: 8, 16, and 32 bits. Each test phase writes a 16-byte data pattern (using its data size) to the first 16 bytes of every 256-byte block of memory in the test range. The 256-byte blocks of memory are aligned to the starting address configuration parameter for the **RAM** test group. The test phase then reads and verifies the 16-byte block using 8-, 16-, and 32-bit access modes.

Command Input:

`162-Diag>`**RAM PERM**

or:

`162-Diag>`**SRAM PERM**

Response/Messages:

Note that in all responses shown below, the response "`RAM` " is `RAM` or `SRAM`, depending upon which test set is being performed.

After the command has been issued, the following line is printed:

```
RAM     PERM: Permutations......................... Running --->
```

If all parts of the test are completed correctly, then the test passes.

```
RAM     PERM: Permutations......................... Running ---> PASSED
```

If the test fails, then the display appears as follows:

```
RAM     PERM: Permutations......................... Running ---> FAILED

Data Miscompare Error:
Address =________, Expected =________, Actual =________
```

Quick Write/Read - QUIK

Verifies specified memory locations. Each pass of this test fills the test range with a data pattern by writing the current data pattern to each memory location from a local variable and reading it back into that same register. This test is coded to use only 32-bit data entities.

The local variable is verified to be unchanged only after the write pass through the test range. This test uses a first pass data pattern of 0, and $FFFFFFFF for the second pass.

Command Input:

```
162-Diag>RAM QUIK
```

or:

```
162-Diag>SRAM QUIK
```

Response/Messages:

Note that in all responses shown below, the response "RAM " is RAM or SRAM, depending upon which test set is being performed.

After the command has been issued, the following line is printed:

```
RAM     QUIK: Quick Write/Read...................... Running --->
```

If all parts of the test are completed correctly, then the test passes.

```
RAM     QUIK: Quick Write/Read...................... Running ---> PASSED
```

If the test fails, then the display appears as follows:

```
RAM     QUIK: Quick Write/Read...................... Running ---> FAILED

Data Miscompare Error: Expected =________, Actual =________
```

Memory Refresh Testing - REF

Verifies specified memory locations after a refresh wait cycle. The memory range and address increment is specified by the RAM test directory configuration parameters. (Refer to the CF command.)

First, the real time clock is checked to see if it is functioning properly. Second, each memory location to be tested has the data portion verified by writing/verifying all zeros, and all ones. Next a data pattern is written to the test location. After all the data patterns are filled for all test locations, a refresh wait cycle is executed. After the wait cycle, the data is read, and if the previously entered data pattern does not match the data pattern read in, a failure occurs. If the data patterns match, then the test is passed.

Command Input:

`162-Diag>`**RAM REF**

or:

`162-Diag>`**SRAM REF**

Response/Messages:

Note that in all responses shown below, the response "`RAM` " is `RAM` or `SRAM`, depending upon which test set is being performed.

After the command has been issued, the following line is printed:

```
RAM       REF:  Memory Refresh.................. Running --->
```

If all parts of the test are completed correctly, then the test passes.

```
RAM       REF:  Memory Refresh.................. Running ---> PASSED
```

If any part of the test fails, then the display appears as follows:

```
RAM       REF:  Memory Refresh.................. Running ---> FAILED
(error message)
```

Here, (error message) is one of the following:

If the real time clock is not functioning properly, one of the following is printed:

```
RAM/REF Test Failure Data:
```

```
RTC is stopped, invoke SET command.
```

or:

```
RAM/REF Test Failure Data:
RTC is in write mode, invoke SET command.
```

or:

```
RAM/REF Test Failure Data:
RTC is in read mode, invoke SET command.
```

If a data verification error occurs before the refresh wait cycle:

```
RAM/REF Test Failure Data:

Immediate Data Miscompare Error:
Address =________, Expected =________, Actual =________
```

If a data verification error occurs following the refresh wait cycle:

```
RAM/REF Test Failure Data:

Unrefreshed Data Miscompare Error:
Address =________, Expected =________, Actual =________
```

Random Data - RNDM

Verifies addressing of memory in the range specified by the configuration parameters for the **RAM** test group. The test proceeds as follows:

1. A random pattern is written throughout the test block.
2. The random pattern complemented is written throughout the test block.
3. The complemented pattern is verified.
4. The random pattern is rewritten throughout the test block.
5. The random pattern is verified.

This test is coded to use only 32-bit data entities. Each time this test is executed, the random seed in the **RAM** test group configuration parameters is post incremented by 1.

Command Input:

`162-Diag>`**RAM RNDM**

or:

`162-Diag>`**SRAM RNDM**

Response/Messages:

Note that in all responses shown below, the response "`RAM `" is `RAM` or `SRAM`, depending upon which test set is being performed.

After the command has been issued, the following line is printed:

```
RAM     RNDM: Random Data........................... Running --->
```

If all parts of the test are completed correctly, then the test passes.

```
RAM     RNDM: Random Data........................... Running ---> PASSED
```

If the test fails, then the display appears as follows:

```
RAM     RNDM: Random Data........................... Running ---> FAILED

Data Miscompare Error:
Address =________, Expected =________, Actual =________
```

MK48T0x (RTC) Tests

These tests check the BBRAM, SRAM, and clock portions of the MK48T08 Real Time Clock (RTC) chips.

Entering **RTC** without parameters causes all **RTC** tests to execute in the order shown in the table below.

To run an individual test, add that test name to the **RTC** command. The individual tests are described in alphabetical order on the following pages.

Table 3-3. RTC Test Group

Mnemonic	Description
CLK	Clock Function
RAM	Battery Backed-Up SRAM
ADR	BBRAM Addressing

BBRAM Addressing - ADR

Assures proper addressability of the MK48T0x BBRAM. The algorithm used is to fill the BBRAM with data pattern "a", a single address line of the MK48T0x is set to one, and pattern "b" is written to the resultant address. All other locations in the BBRAM are checked to ensure that they were not affected by this write. The "a" pattern is then restored to the resultant address. All address lines connected to the MK48T0x are tested in this manner.

Since this test overwrites all memory locations in the BBRAM, the BBRAM contents are saved in debugger system memory prior to writing the BBRAM. The RTC test group features a configuration parameter which overrides automatic restoration of the BBRAM contents. The default for this parameter is to restore BBRAM contents upon test completion.

Command Input:

```
162-Diag>RTC ADR
```

Response/Messages:

After the command has been issued, the following line is printed:

```
RTC ADR: MK48T0x RAM Addressing................ Running --->
```

If all parts of the test are completed correctly, then the test passes.

```
RTC ADR: MK48T0x RAM Addressing................ Running ---> PASSED
```

If any part of the test fails, then the display appears as follows:

```
RTC ADR: MK48T0x RAM Addressing................ Running ---> FAILED
(error message)
```

Here, (error message) is one of the following:

If debugger system memory cannot be allocated for use as a save area for the BBRAM contents:

```
RAM allocate
memc.next=________ memc.size=________
```

If the BBRAM cannot be initialized with pattern "a":

```
Data Verify Error: Address =________, Expected =__, Actual =__
Memory initialization error
```

If a pattern "b" write affects any BBRAM location other than the resultant address:

```
Data Verify Error: Address =________, Expected =__, Actual =__
Memory addressing error - wrote __ to ________
```

Clock Function - CLK

Verifies the functionality of the Real Time Clock (RTC). This test does not check clock accuracy.

This test requires approximately nine seconds to run. At the conclusion of the test, nine seconds are added to the clock time to compensate for the test delay. Because the clock can only be set to the nearest second, this test may induce ± one second of error into the clock time.

Command Input:

`162-Diag>`**RTC CLK**

Response/Messages:

After the command has been issued, the following line is printed:

```
RTC     CLK: MK48T0x Real Time Clock................ Running --->
```

If all parts of the test are completed correctly, then the test passes.

```
RTC     CLK: MK48T0x Real Time Clock................ Running ---> PASSED
```

If the test fails, then the display appears as follows:

```
RTC      CLK: MK48T0x Real Time Clock................ Running ---> FAILED
(error message)
```

Here, (error message) is one of the following:

If the check for low battery fails.

```
RTC low battery
```

Note

The Low Battery test only assures Battery OK if the MK48T02 (used on other boards) has not been written since powerup. The Battery test is performed here in case the debugger currently in use does not perform a Low Battery test on powerup. Although the MK48T08 does not support the internal battery voltage check (BOK), the BOK flag status check algorithm is performed by this test on all parts.

The RTC time registers are configured for constant updating by the clock internal counters. The seconds register is read initially and then monitored (read) to verify that the seconds value changes. A predetermined number of reads are made of the seconds register. If the predetermined number of reads are made before the seconds register changed, the following message is printed:

```
RTC not running
```

The RTC time registers are configured for reading. A predetermined number of MPU "do nothing" loops are executed. If the seconds register changes before the full count of MPU loops is executed, the following message is printed:

```
RTC did not freeze for reading
```

If the real time clock registers fail the data pattern test:

```
Data Miscompare Error:
Address =________, Expected =________, Actual =________
```

The following message indicates a programming error and should never be seen by the diagnostics user:

```
WARNING -- Real Time Clock NOT compensated for test delay.
```

Battery Backed-Up SRAM - RAM

Performs a data test on each SRAM location of the Mostek MK48T08 "Zeropower" RAM. RAM contents are unchanged upon completion of test, regardless of pass or fail test return status. This test is coded to test only byte data entities. The test proceeds as follows:

1. For each of the following patterns: $1, $3, $7, $f, $1f, $3f, and $7f:
2. For each valid byte of the "Zeropower RAM":
3. Write and verify the current data test pattern.
4. Write and verify the complement of the current data test pattern.

Command Input:

`162-Diag>`**RTC RAM**

Response/Messages:

After the command has been issued, the following line is printed:

```
RTC     RAM: MK48T0x Battery Backed Up RAM.......... Running --->
```

If all parts of the test are completed correctly, then the test passes.

```
RTC     RAM: MK48T0x Battery Backed Up RAM.......... Running ---> PASSED
```

If the test fails, then the display appears as follows:

```
RTC      RAM: MK48T0x Battery Backed Up RAM.......... Running ---> FAILED
(error message)
```

Here, (error message) is the following:

```
Data Miscompare Error:
Address =________, Expected =________, Actual =________
```

Memory Controller Chip (MCC) Tests

This section describes the Memory Controller Chip (MCC) tests.

The MCC ASIC is one of three ASICs that are part of the MVME162 hardware set. The MCC is designed to operate synchronously with the MC68040 local bus clock at 25MHZ or 33MHZ.

Entering **MCC** without parameters causes all MCC tests to execute in the order shown in the next table.

To run an individual test, add that test name to the **MCC** command. The individual tests are described in alphabetical order on the following pages. The error message displays following the explanation of an MCC test pertain to the test being discussed.

Table 3-4. MCC Test Group

Mnemonic	Description
ACCESSA	Device Access
ACCESSB	Register Access
TMR1A	Timer 1 Counter
TMR1B	Timer 1 Free-Run
TMR1C	Timer 1 Clear On Compare
TMR1D	Timer 1 Overflow Counter
TMR1E	Timer 1 Interrupts
TMR2A	Timer 2 Counter
TMR2B	Timer 2 Free-Run
TMR2D	Timer 2 Overflow Counter
TMR2E	Timer 2 Interrupts
TMR3A	Timer 2 Counter
TMR3B	Timer 2 Free-Run
TMR3C	Timer 2 Clear On Compare
TMR3D	Timer 2 Overflow Counter
TMR3E	Timer 2 Interrupts
TMR4A	Timer 2 Counter
TMR4B	Timer 2 Free-Run
TMR4C	Timer 2 Clear On Compare
TMR4D	Timer 2 Overflow Counter
TMR4E	Timer 2 Interrupts

Table 3-4. MCC Test Group (Continued)

Mnemonic	Description
ADJ	Prescaler Clock Adjust
PCLK	Prescaler Clock
MPUCS	MPU Clock Speed
RFRSH	DRAM Refresh Timing
FAST	FAST Bit
WDTMRA	Watchdog Timer Counter
WDTMRB	Watchdog Timer Board Fail
VBR	Vector Base Register
Executed only when specified:	
WDTMRC	Watchdog Timer Local Reset

Device Access - ACCESSA

Verifies that the MCC register set can be accessed (read) on byte, word, and long word boundaries (where applicable). No attempt is made to verify the contents of the registers.

Command Input:

`162-Diag>`**MCC ACCESSA**

Response/Messages:

After the command has been issued, the following line is printed:

```
MCC     ACCESSA: Device Access....................... Running --->
```

If all parts of the test are completed correctly, then the test passes.

```
MCC      ACCESSA: Device Access....................... Running --->
PASSED
```

If any part of the test fails, then the display appears as follows:

```
MCC      ACCESSA: Device Access....................... Running --->
FAILED
MCC/ACCESSA Test Failure Data:
(error message)
```

Here, (error message) is one of the following:

```
            Bus Error Information:
                           Address ________
                              Data ________
                       Access Size __
                       Access Type _
                Address Space Code _
                     Vector Number ___



            Unsolicited Exception:
                   Program Counter ________
                     Vector Number __
                       Access Size ___
                   Status Register ____
                   Interrupt Level _
```

Register Access - ACCESSB

Checks the device data lines by successive writes and reads to all tick timers compare and counter registers. The test walks a 1 bit through a field of zeros and walks a 0 bit through a field of ones.

Command Input:

```
162-Diag>MCC ACCESSB
```

Responses/Messages:

After the command has been issued, the following line is printed:

```
MCC    ACCESSB: Register Access...................... Running --->
```

If all parts of the test are completed correctly, then the test passes.

```
MCC      ACCESSB: Register Access..................... Running --->
PASSED
```

If any failures occur, the following is displayed (more descriptive text then follows):

```
MCC      ACCESSB: Register Access..................... Running --->
FAILED
MCC/ACCESSB Test Failure Data:
(error message)
```

Here, (error message) is one of the following:

```
Register did not clear
Address =________, Expected =________, Actual =________
```

```
Register access error
Address =________, Expected =________, Actual =________
```

Prescaler Clock Adjust - ADJ

Verifies that the Prescaler Clock Adjust Register can vary the period of the Tick Timer input clock. This is accomplished by setting the Clock Adjust Register to zero and allowing Tick Timer 1 to free-run for a small software delay to establish a reference count. Next a 1 is walked through the Clock Adjust Register and the timer is allowed to run for the same delay period; the resulting count should be greater than the last (previous) count.

Command Input:

`162-Diag>`**MCC ADJ**

Responses/Messages:

After the command has been issued, the following line is printed:

```
MCC     ADJ: Prescaler Clock Adjust................. Running --->
```

If all parts of the test are completed correctly, then the test passes.

```
MCC     ADJ: Prescaler Clock Adjust................. Running ---> PASSED
```

If any failures occur, the following is displayed (more descriptive text then follows):

```
MCC      ADJ: Prescaler Clock Adjust................. Running ---> FAILED
MCC/ADJ Test Failure Data:
(error message)
```

Here, (error message) is one of the following:

```
Prescaler Clock Adjust Register not initialized
Register Address =________, should not be zero
```

```
Clock Adjust did not vary tick period correctly
Register Address =________, Adjust Value =__
Test Count     =________, should be greater than
Previous Count =________
```

FAST Bit - FAST

Verifies the FAST/SLOW access time to the BBRAM. This is accomplished by using Tick Timer #1. The tick timer is first used to time 4000 accesses to the BBRAM with the FAST bit set. Then the FAST bit is cleared and the tick timer is used to time 4000 accesses to the BBRAM. The count measured when the FAST bit is set should be less than the count measured when the FAST bit is cleared.

Command Input:

`162-Diag>`**MCC FAST**

Responses/Messages:

After the command has been issued, the following line is printed:

```
MCC     FAST: `FAST' Bit............................ Running --->
```

If all parts of the test are completed correctly, then the test passes.

```
MCC     FAST: `FAST' Bit............................ Running ---> PASSED
```

If any failures occur, the following is displayed (more descriptive text then follows):

```
MCC     FAST: `FAST' Bit............................ Running ---> FAILED
```

```
MCC/FAST Test Failure Data:
(error message)
```

Here, (error message) is one of the following:

```
`FAST' bit did not vary access time correctly
Fast access count =________, Slow access count =________
Fast count should be less than Slow count
```

MPU Clock Speed - MPUCS

Verifies that the calculated MPU clock speed matches both the version register of the MCC and the MCC prescaler initialized value. The MPU clock speed calculation is done by using the RTC (MK48T08) and Tick Timer #1.

Command Input:

`162-Diag>`**MCC MPUCS**

Responses/Messages:

After the command has been issued, the following line is printed:

```
MCC     MPUCS: MPU Clock Speed...................... Running --->
```

If all parts of the test are completed correctly, then the test passes.

```
MCC     MPUCS: MPU Clock Speed...................... Running ---> PASSED
```

If any failures occur, the following is displayed (more descriptive text then follows):

```
MCC     MPUCS: MPU Clock Speed...................... Running ---> FAILED

MCC/MPUCS Test Failure Data:
(error message)
```

Here, (error message) is one of the following:

```
MPU Clock Speed Calculation failed, RTC not operating

MPU Clock Speed Calculation does not match the MCC Version Register
Calculated MPU Clock Speed =__MHZ (&________HZ), Version Register =__MHZ

MPU Clock Speed Calculation does not match the MCC Prescaler Register
Calculated MPU Clock Speed =__MHZ (&________HZ), Prescaler Register =_MHZ

Unknown Prescaler Adjust Value =__
```

Prescaler Clock - PCLK

Verifies the accuracy of the Prescaler Clock. This is accomplished by using a constant time source, in this case the MK48T08 RTC, and Tick Timer #1. First, the constant time source is verified for operation. Then the tick timer is initialized and the constant time source is brought to a whole second interval. The tick timer is started and the constant time source is polled for the next second to roll over while the tick timer is free-running; upon roll over the tick timer is stopped. The elapsed tick timer count is verified; acceptance of this count allows for a plus or minus 0.1 percent tolerance.

Command Input:

`162-Diag>`**MCC PCLK**

Responses/Messages:

After the command has been issued, the following line is printed:

```
MCC     PCLK: Prescaler Clock....................... Running --->
```

If all parts of the test are completed correctly, then the test passes.

```
MCC     PCLK: Prescaler Clock....................... Running ---> PASSED
```

If any failures occur, the following is displayed (more descriptive text then follows):

```
MCC     PCLK: Prescaler Clock....................... Running ---> FAILED
```

```
MCC/PCLK Test Failure Data:
(error message)
```

Here, (error message) is one of the following:

```
Unknown Prescaler Adjust Value =__
```

```
RTC seconds register didn't increment
```

```
Timer count register greater/less than expected
Address =________, Expected =________, Actual =________
```

DRAM Refresh Timing - DRAM

Verifies that when the refresh rate is changed via the Bus Clock Register, the total time of access to the Dynamic RAM (DRAM) array also changes. Accesses to the DRAM array should be held off until the refresh cycle is complete. Tick Timer #1 in the MCC is used to make the necessary elapsed time measurements. The refresh timing logic is tested for both the maximum ($01) and the minimum ($00) refresh periods. The elapsed time of maximum should be larger than elapsed time of minimum.

Command Input:

```
162-Diag>MCC RFRSH
```

Responses/Messages:

After the command has been issued, the following line is printed:

```
MCC     RFRSH: DRAM Refresh Timing.................. Running --->
```

If all parts of the test are completed correctly, then the test passes.

```
MCC     RFRSH: DRAM Refresh Timing.................. Running ---> PASSED
```

If any failures occur, the following is displayed (more descriptive text then follows):

```
MCC     RFRSH: DRAM Refresh Timing.................. Running ---> FAILED

MCC/RFRSH Test Failure Data:
(error message)
```

Here, (error message) is one of the following:

```
Refresh Period did not vary as expected
Elapsed Time of MAXIMUM should be larger than Elapsed Time of MINIMUM
MAXIMUM Period Value =__, Elapsed Time =________
MINIMUM Period Value =__, Elapsed Time =________
```

Timer Counters - TMR*n*A

Verify that each Tick Timer's Counters are operational. These tests have three parts to them.

1. The Tick Timer counter register is verified for data write/read operability. Both a 1 and 0 bit are walked through the 32-bit register and verified.
2. The Tick Timer counter register is initialized to zero and the counter is enabled. The test verifies that the counter register becomes non-zero (increments).
3. The the Tick Timer counter register is initialized to a predetermined count (i.e., $00000000, $00000001, $00000003, $00000007, ..., $7FFFFFFF, $FFFFFFFF) and then enabled. The test waits for the contents of the counter register to be greater than the initialization count.

Command Input:

```
162-Diag>MCC TMR1A
162-Diag>MCC TMR2A
162-Diag>MCC TMR3A
162-Diag>MCC TMR4A
```

Responses/Messages:

Note that the responses indicate which timer you are testing. These examples illustrate the responses for Tick Timer 1.

After the command has been issued, the following line is printed:

```
MCC     TMR1A: Timer 1 Counter...................... Running --->
```

If all parts of the test are completed correctly, then the test passes.

```
MCC     TMR1A: Timer 1 Counter...................... Running ---> PASSED
```

If any failures occur, the following is displayed (more descriptive text then follows):

```
MCC     TMR1A: Timer 1 Counter...................... Running ---> FAILED
```

```
MCC/TMR1A Test Failure Data:
(error message)
```

Here, (error message) is one of the following:

```
Register did not clear
Address =________, Expected =________, Actual =________

Register access error
Address =________, Expected =________, Actual =________

Counter did not increment
Address =________, Expected =________, Actual =________

Timeout waiting for Counter to increment
Address =________, Expected =________, Actual =________

Timeout waiting for Counter to roll over
Address =________, Expected =________, Actual =________
```

Timer Free-Run - TMR*n*B

Verify that the Tick Timers' Compare Registers are operational. These tests have two parts to them.

1. The Tick Timer's compare register is verified for data write/read operability. Both a 1 and 0 bit are walked through the 32-bit register and verified.
2. The Tick Timer's counter and compare registers are initialized to a predetermined count (i.e., $00000000, $00000001, $00000003, $00000007, ..., $7FFFFFFF, $FFFFFFFF). Then the counter is enabled and the test waits for the contents of the counter register to exceed the compare register's initial contents. This also verifies that the counter register will not clear when the compare count is met.

Command Input:

```
162-Diag>MCC TMR1B
162-Diag>MCC TMR2B
162-Diag>MCC TMR3B
162-Diag>MCC TMR4B
```

Responses/Messages:

Note that the responses indicate which timer you are testing. These examples illustrate the responses for Tick Timer 1.

After the command has been issued, the following line is printed:

```
MCC     TMR1B: Timer 1 Free-Run..................... Running --->
```

If all parts of the test are completed correctly, then the test passes.

```
MCC     TMR1B: Timer 1 Free-Run..................... Running ---> PASSED
```

If any failures occur, the following is displayed (more descriptive text then follows):

```
MCC     TMR1B: Timer 1 Free-Run..................... Running ---> FAILED
```

```
MCC/TMR1B Test Failure Data:
(error message)
```

Here, (error message) is one of the following:

```
Register did not clear
Address =________, Expected =________, Actual =________

Register access error
Address =________, Expected =________, Actual =________

Timeout waiting for Count to exceed Compare
Address =________, Expected =________, Actual =________
```

Timer Clear on Compare - TMR*n*C

Verify the Tick Timers' Clear on Compare functions. This is accomplished by initializing the compare and count registers and then enabling the timer to run until the software times-out or the counter exceeds the compare (error condition). Both the counter and compare registers on the Tick Timer are initialized to a predetermined count (i.e. $00000000, $00000001, $00000003, $00000007, ..., $7FFFFFFF, $FFFFFFFF). Then the counter is enabled and the test waits for a small amount of time (software delay) or for the contents of the counter register to exceed the compare register's initial contents.

Command Input:

```
162-Diag>MCC TMR1C
162-Diag>MCC TMR2C
162-Diag>MCC TMR3C
162-Diag>MCC TMR4C
```

Responses/Messages:

Note that the responses indicate which timer you are testing. These examples illustrate the responses for Tick Timer 1.

After the command has been issued, the following line is printed:

```
MCC     TMR1C: Timer 1 Clear on Compare............. Running --->
```

If all parts of the test are completed correctly, then the test passes.

```
MCC     TMR1C: Timer 1 Clear on Compare............. Running ---> PASSED
```

If any failures occur, the following is displayed (more descriptive text then follows):

```
MCC     TMR1C: Timer 1 Clear on Compare............. Running ---> FAILED
```

```
MCC/TMR1C Test Failure Data:
(error message)
```

Here, (error message) is one of the following:

```
Count did not zero on Compare
Address =________, Expected =________, Actual =________
```

Timer Overflow Counter - TMR*n*D

Verify the Tick Timers' Overflow Counter functions. The test has four parts.

1. The test verifies that the overflow counter can be cleared to zero.
2. The test verifies that the overflow counter will increment from $00 to $10.
3. The test verifies the overflow count can be cleared once set (non-zero).
4. The test verifies that the overflow count can increment from $10 to $F0 (increments of $10).

Command Input:

```
162-Diag>MCC TMR1D
162-Diag>MCC TMR2D
162-Diag>MCC TMR3D
162-Diag>MCC TMR4D
```

Responses/Messages: `162-Diag>`**MCC TMR4D**

Note that the responses indicate which timer you are testing. These examples illustrate the responses for Tick Timer 1.

After the command has been issued, the following line is printed:

```
MCC     TMR1D: Timer 1 Overflow Counter............ Running --->
```

If all parts of the test are completed correctly, then the test passes.

```
MCC     TMR1D: Timer 1 Overflow Counter............ Running ---> PASSED
```

If any failures occur, the following is displayed (more descriptive text then follows):

```
MCC     TMR1D: Timer 1 Overflow Counter............ Running ---> FAILED
```

```
MCC/TMR1D Test Failure Data:
(error message)
```

Here, (error message) is one of the following:

```
Overflow Counter did not clear
Address =________, Expected =__, Actual =__
```

```
Overflow Counter did not increment
Address =________, Expected =__, Actual =__
```

```
Timeout waiting for Overflow Counter
Address =________, Expected =__, Actual =__
```

Timer Interrupts - TMR*n*E

Verify that the Tick Timers can generate interrupts and the MPU takes the correct vector. The test verifies that level 0 interrupts will not generate an interrupt, but will set the appropriate status. The test then verifies that all interrupts (1-7) can be generated and received and that the appropriate status is set.

Command Input:

```
162-Diag>MCC TMR1E
162-Diag>MCC TMR2E
162-Diag>MCC TMR3E
162-Diag>MCC TMR4E
```

Responses/Messages:

Note that the responses indicate which timer you are testing. These examples illustrate the responses for Tick Timer 1.

After the command has been issued, the following line is printed:

```
MCC     TMR1E: Timer 1 Interrupts................... Running --->
```

If all parts of the test are completed correctly, then the test passes.

```
MCC     TMR1E: Timer 1 Interrupts................... Running ---> PASSED
```

If any failures occur, the following is displayed (more descriptive text then follows):

```
MCC     TMR1E: Timer 1 Interrupts................... Running ---> FAILED
```

```
MCC/TMR1E Test Failure Data:
(error message)
```

Here, (error message) is one of the following:

```
Interrupt Control Register did not clear
Address =________, Expected =__, Actual =__

Interrupt Enable bit did not set
Address =________, Expected =__, Actual =__

Bus Error Information:
               Address ________
                  Data ________
           Access Size __
           Access Type _
    Address Space Code _
         Vector Number ___


Unsolicited Exception:
       Program Counter ________
         Vector Number __
           Access Size ___
       Status Register ____
       Interrupt Level _

Interrupt Status bit did not set
Status: Expected =__, Actual =__
Vector: Expected =__, Actual =__
State : IRQ Level =_, VBR =__

Incorrect Vector type
Status: Expected =__, Actual =__
Vector: Expected =__, Actual =__
State : IRQ Level =_, VBR =__

Unexpected Vector taken
Status: Expected =__, Actual =__
Vector: Expected =__, Actual =__
State : IRQ Level =_, VBR =__

Incorrect Interrupt Level
Level: Expected =_, Actual =_
State: IRQ Level =_, VBR =__

Interrupt did not occur
Status: Expected =__, Actual =__
Vector: Expected =__, Actual =__
State : IRQ Level =_, VBR =__

Interrupt Status bit did not clear
Address =________, Expected =__, Actual =__
```

Vector Base Register - VBR

Verifies that the MCC's Vector Base Register is operational. First, the register is tested for all possible data patterns. Second, the Vector Base Register is tested for vector direction. Vector bases of $40 to $F0 (increments of $10) are used/tested. Vector direction is accomplished by using the LAN Coprocessor Interrupt Control Register (MCC based) as the interrupt source.

Command Input:

162-Diag>**MCC VBR**

Responses/Messages:

After the command has been issued, the following line is printed:

```
MCC     VBR: Vector Base Register................... Running --->
```

If all parts of the test are completed correctly, then the test passes.

```
MCC     VBR: Vector Base Register................... Running ---> PASSED
```

If any failures occur, the following is displayed (more descriptive text then follows):

```
MCC     VBR: Vector Base Register................... Running ---> FAILED
```

```
MCC/VBR Test Failure Data:
(error message)
```

Here, (error message) is one of the following:

```
Write/Read error on VBR
Address =________, Expected =__, Actual =__
```

```
Unexpected Vector taken
Status: Expected =__, Actual =__
Vector: Expected =__, Actual =__
State : IRQ Level =_, VBR =__
```

```
Interrupt did NOT occur
Expected Vector =__ (&___)
IRQ Level =_, VBR =__, Control/Status Register =__
```

Watchdog Timer Counter - WDTMRA

Verifies that the Watchdog Timer Counter will count and set the correct status (time-out). The time-out status is verified that it can be cleared. The counter is tested for write/read capability of all settings. The counter selection timeouts are only tested with reasonable settings (4 seconds and under) to keep the total test time short.

Command Input:

`162-Diag>`**MCC WDTMRA**

Responses/Messages:

After the command has been issued, the following line is printed:

```
MCC     WDTMRA: Watchdog Timer Counter.............. Running --->
```

If all parts of the test are completed correctly, then the test passes.

```
MCC     WDTMRA: Watchdog Timer Counter.............. Running ---> PASSED
```

If any failures occur, the following is displayed (more descriptive text then follows):

```
MCC     WDTMRA: Watchdog Timer Counter.............. Running ---> FAILED
```

```
MCC/WDTMRA Test Failure Data:
(error message)
```

Here, (error message) is one of the following:

```
Register Write/Read Error
Address =________, Expected =__, Actual =__
```

```
Software Time-Out occurred while waiting for Watchdog Time-Out Status
Time-Out Selection Code =__
```

```
Current Time-Out Selection Code Count was NOT greater than the Previous
Current Count =________, Previous Count =________
```

```
Watchdog Time-Out Status Bit did NOT clear
Watchdog Timer Control Register =__
Time-Out Selection Code         =__
```

Watchdog Timer Board Fail - WDTMRB

Verifies that the Watchdog Timer will set the Board Fail indicator (FAIL LED and VMEbus SYSFAIL*) when a time-out occurs. The test also verifies that the Board Fail Signal and Indicator can be toggled, both from the MCC and the VMECHIP2.

Command Input:

```
162-Diag>MCC WDTMRB
```

Responses/Messages:

After the command has been issued, the following line is printed:

```
MCC     WDTMRB: Watchdog Timer Board Fail........... Running --->
```

If all parts of the test are completed correctly, then the test passes.

```
MCC     WDTMRB: Watchdog Timer Board Fail........... Running ---> PASSED
```

If any failures occur, the following is displayed (more descriptive text then follows):

```
MCC     WDTMRB: Watchdog Timer Board Fail........... Running ---> FAILED

MCC/WDTMRB Test Failure Data:
(error message)
```

Here, (error message) is one of the following:

```
Board Fail Signal/Indicator is NOT clear
Reset Switch Control Register =__
(Board Fail Signal is driven (negated) via MCC)

Board Fail Signal/Indicator is NOT clear
Reset Switch Control Register =__
(Board Fail Signal is driven (negated) via VMEC2)

Board Fail Signal/Indicator is NOT set
Reset Switch Control Register =__
(Board Fail Signal is driven (asserted) via MCC)

Board Fail Signal/Indicator is NOT set
Reset Switch Control Register =__
(Board Fail Signal is driven (asserted) via VMEC2)

Board Fail Signal/Indicator is NOT set
Reset Switch Control Register =__
(Board Fail Signal is driven (asserted) via MCC Watchdog Timer)
```

```
Board Fail Signal/Indicator is NOT clear
Reset Switch Control Register =__
(Board Fail Signal is driven (negated) via MCC Watchdog Timer)
```

Watchdog Timer Local Reset - WDTMRC

Verifies that the Watchdog Timer will generate a local reset upon timing out. If the test returns, it is considered a failure and an appropriate error message is displayed/logged.

Note that this test does not execute when the MCC test group is executed (MCC with no arguments). This test is supplied only for diagnostic purposes.

Command Input:

```
162-Diag>MCC WDTMRC
```

Responses/Messages:

After the command has been issued, the following line is printed:

```
MCC     WDTMRC: Watchdog Timer Local Reset.......... Running --->
```

If all parts of the test are completed correctly, then the test passes.

```
MCC     WDTMRC: Watchdog Timer Local Reset.......... Running ---> PASSED
```

If any failures occur, the following is displayed (more descriptive text then follows):

```
MCC     WDTMRC: Watchdog Timer Local Reset.......... Running ---> FAILED
```

```
MCC/WDTMRC Test Failure Data:
(error message)
```

Here, (error message) is one of the following:

```
Register Write/Read Error
Address =________, Expected =__, Actual =__
```

```
Watchdog Timer did NOT generate a Local Bus Reset
```

Additional MCC Test Group Messages

The following error messages (and descriptions for each) may apply to any or all of the tests within the MCC test group.

```
Bus Error Information:
               Address ________
                  Data ________
           Access Size __
           Access Type _
    Address Space Code _
         Vector Number ___

Unsolicited Exception:
       Program Counter ________
         Vector Number __
           Access Size ___
       Status Register ____
       Interrupt Level _
```

ECC Memory Board (MCECC) Tests

This section describes the individual MCECC memory tests.

Entering **MCECC** without parameters causes all MCECC tests to execute in the order shown in the table below.

To run an individual test, add that test name to the **MCECC** command. The individual tests are described in alphabetical order on the following pages.

Table 3-5. MCECC Test Group

Mnemonic	Description
CBIT	Check-Bit DRAM
SCRUB	Scrubbing
SBE	Single-Bit-Error
MBE	Multi-Bit-Error
EXCPTN	Exceptions

Configuration of some parameters that these tests use may be accomplished through the use of the **CF** command:

```
162-Diag>CF MCECC
```

The first question asked is:

```
Inhibit restore of ECC registers upon test failure (y/n) =n ?
```

This allows someone trying to debug a problem with an MCECC memory board to maintain the state of all the ASIC's registers after a failure occurs. Otherwise, the registers will be "cleaned up" before the diagnostic exits.

The next question is:

```
Verbose messages during execution (y/n) =n ?
```

This allows you to request that extra display output be generated on the console line, that shows what portion of the test is being executed. Because of the large size of these memory boards, some of these tests can take many minutes to execute. Having the extra output can help to assure you that the test is indeed still running.

The next question is:

```
Override default starting/ending addresses (y/n) =n ?
```

This allows you to override the default address ranges for testing, on a per board basis. The default answer "n" means that the MCECC diagnostics check the environment, and test all possible memory on every MCECC board found in the system.

Following this line, the starting and ending addresses for each memory board are displayed:

```
Starting address, 1st|2nd memory board (hex, 0 - 08000000) =00000000 ?
Ending address, 1st|2nd memory board   (hex, 0 - 08000000) =00000000 ?
```

These addresses are relative to the particular board only. Each board address begins at zero, despite where it might be configured in the computer's memory map. If a system is configured with two 32MB ECC memory boards, then for purposes of the configuration parameters, each board starts at address 0, and ends at 02000000.

Check-Bit DRAM - CBIT

This test verifies the operation of the check-bit RAM. The test uses the address as the data in the first word, the complement of the address in the second word, and swapped nybbles in the third word. This pattern continues all through the checkbit memory. When complete, this process is repeated two more times, but the order of the functions for generating checkbit data are rotated until each word has used each of the three types of data-generating functions.

The SBC ECC memory boards are comprised of two MCECC ASICs, and DRAM connected to each ASIC. The ASICs have a control bit that may be set, to allow direct reading/writing of checkbit memory. In this test, that bit is set, and causes each of the two checkbit words to appear in separate bytes of the data word (bits 8-15 = lower MCECC, bits 24-31 = upper MCECC). The test data is then masked to 8 bits, and copied into bits 8-15 and 24-31. All of check-bit RAM is written in one pass, followed by a verification pass of all of RAM.

Command Input:

`162-Diag>`**MCECC CBIT**

Response/Messages:

If "verbose" execution has been configured with the **CF** command, something will be printed on the status line to let you know that the test is still running. The first message indicates that the ECC checkbits are being initialized. The format of the status update is: ________ *x#p* where the "________" portion is the current address being accessed, and the "*x*" is replaced by either "w" or "r" depending on whether the current pass through memory is write or read. The "#" indicates the memory board number being tested, and the "*p*" is replaced with one of "a", "b", or "c", according to which pass of the addressability test is being executed.

```
ECC   CBIT: ECC Check-Bit DRAM................. Running ---> bd # init
ECC    CBIT: ECC Check-Bit DRAM................. Running ---> ________
x#p
ECC   CBIT: ECC Check-Bit DRAM................. Running ---> PASSED
```

If an error is detected during the operation of the test, one of the following messages is displayed:

Failures due to the scrubber during checkbit initialization:

```
Timed out waiting for scrubber to start, bd #_ (status __)
Timed out waiting for scrubber to stop, bd #_ (status __)
```

Failures in checkbit memory:

```
At:  ________, read:  ________, should be:  ________, (lower MCECC)
At:  ________, read:  ________, should be:  ________, (upper MCECC)
```

The test ends with:

```
      ECC      CBIT: ECC Check-Bit DRAM................. Running ---> FAILED
```

Exceptions - EXCPTN

This test verifies the operation of the MCECC's capability to generate interrupts, or bus errors on detecting a memory error. This test plants errors in memory, enables either the interrupt or bus-error, and then reads the "faulty" memory location. The proper exception and status is tested, and if received, the test passes.

Command Input:

`162-Diag>`**MCECC EXCPTN**

Response/Messages:

If an error is detected during the operation of the test, one of the following messages is displayed:

Failures due to the scrubber during checkbit initialization:

```
Timed out waiting for scrubber to start, bd #_ (status __)
Timed out waiting for scrubber to stop, bd #_ (status __)
```

The test ends with:

```
ECC      EXCPTN: ECC Exceptions.................. Running ---> FAILED
```

Multi-Bit-Error - MBE

This function tests the ECC board's ability to detect multi-bit-errors. It fills a memory area with random data containing a "multi-bit-error" in each word. All of the tested memory area is then verified with error correction enabled, so that the data errors will be detected during the read operation.

Command Input:

```
162-Diag>MBE
```

Response/Messages:

If "verbose" execution has been configured with the **CF** command, something will be printed on the status line to let you know that the test is still running. The first message indicates that the ECC checkbits are being initialized. Next comes a test of the error-logger which displays the `errlog` message. Following this, the multi-bit-error test begins and displays the `mbe` message.

```
ECC    MBE: ECC Multi-Bit-Error................. Running ---> bd # init
ECC     MBE: ECC Multi-Bit-Error................. Running ---> bd # errlog
ECC    MBE: ECC Multi-Bit-Error................. Running ---> bd # mbe

ECC    MBE: ECC Multi-Bit-Error................. Running ---> PASSED
```

If an error is detected during the operation of the test, one of the following messages is displayed:

Failures due to the scrubber during checkbit initialization:

```
Timed out waiting for scrubber to start, bd #_ (status __)
Timed out waiting for scrubber to stop, bd #_ (status __)
```

Failures from the error-logger test:

```
errlog: logger didn't indicate an error:
        bd #_, addr ________, read ________, actual ________
errlog: logger didn't indicate error-on-read, bd #_, addr ________
errlog: logger error address wrong: ________, actual: ________, board #_
```

Errors due to double-bit-errors not being detected properly:

```
mbe: logger didn't indicate an error:
     bd #_, addr ________, read ________, actual ________
mbe: logger didn't indicate error-on-read, bd #_, addr ________
mbe: logger didn't indicate error was multi-bit-error:
     bd #_, addr ________, read ________, actual ________, logger __
mbe: logger error address wrong: ________, actual: ________, board #_
```

The test ends with:

```
ECC      MBE: ECC Multi-Bit-Error................. Running ---> FAILED
```

Single-Bit-Error - SBE

This function tests the ECC board's ability to correct single-bit-errors. It fills a memory area with random data containing a "single-bit-error" in each word. All of the tested memory area is then verified with error correction enabled, so that the data will be "corrected" during the read operation.

Command Input:

`162-Diag>`**MCECC SBE**

Response/Messages:

If "verbose" execution has been configured with the **CF** command, something will be printed on the status line to let you know that the test is still running. The first message indicates that the ECC checkbits are being initialized. The format of the status update is: ________ *x*# where the "________" portion is the current address being accessed, the "*x*" is replaced by either "`w`" or "`r`" depending on whether the current pass through memory is write/read. The "#" indicates the memory board number being tested.

```
ECC   SBE: ECC Single-Bit-Error................ Running ---> bd # init
ECC   SBE: ECC Single-Bit-Error................ Running ---> ________ x#

ECC   SBE: ECC Single-Bit-Error................ Running ---> PASSED
```

If an error is detected during the operation of the test, one of the following messages is displayed:

Failures due to the scrubber during checkbit initialization:

```
Timed out waiting for scrubber to start, bd #_ (status __)
Timed out waiting for scrubber to stop, bd #_ (status __)
```

Errors due to single-bit-errors not being corrected properly:

```
 Address=________, Expected=________, Actual=________
```

The test ends with:

```
ECC      SBE: ECC Single-Bit-Error................ Running ---> FAILED
```

Scrubbing - SCRUB

This function tests refresh "scrubbing" of errors from DRAM. It checks the ECC memory board's capability to correct single-bit-errors during normal DRAM refresh cycles. During its operation, the diagnostic displays the current memory board number that it is working on. When the fast-refresh mode is selected, "wait" is displayed, indicating that the test is waiting long enough for fast-refresh to get to every memory location on the board at least once.

Command Input:

`162-Diag>`**MCECC SCRUB**

Response/Messages:

A short message is printed during the test's progress on the "running" line printed by CTMI, to indicate what "phase" the test is in, and what board it is currently running on. The test begins with an ECC memory initialization, and displays the `init` message. Next comes a test of the error-logger which displays the `errlog` message. Errors are then planted in memory, and the first scrub pass runs with the message `scrub 1` being displayed. The memory is then tested with the error-logger. Finally, another pass of the scrubber is run, and displays the `scrub 2` message. This scrub pass is then checked for zero errors. Finally, if all went well, `PASSED` is displayed.

```
ECC   SCRUB: ECC Scrubbing..................... Running ---> bd # init
ECC    SCRUB: ECC Scrubbing..................... Running ---> bd # errlog
ECC    SCRUB: ECC Scrubbing..................... Running ---> bd # scrub 1
ECC   SCRUB: ECC Scrubbing..................... Running ---> bd # scrub 2

ECC   SCRUB: ECC Scrubbing..................... Running ---> PASSED
```

If an error is detected during the operation of the test, one of the following messages is displayed:

Failures due to the scrubber during checkbit initialization:

```
Timed out waiting for scrubber to start, bd #_ (status __)
Timed out waiting for scrubber to stop, bd #_ (status __)
```

Failures from the error-logger test:

```
errlog: logger didn't indicate an error:
        bd #_, addr ________, read ________, actual ________
errlog: logger didn't indicate error-on-read, bd #_, addr ________
errlog: logger error address wrong: ________, actual: ________, board #_
```

Possible first pass scrubbing failure:

```
Timed out waiting for scrubber to start, bd #_ (status __)
Timed out waiting for scrubber to stop, bd #_ (status __)
single-bit-error at ________ found after scrubbing RAM
multi-bit-error at ________ found after scrubbing RAM
```

Possible second pass scrubbing failure:

```
Timed out waiting for scrubber to start, bd #_ (status __)
Timed out waiting for scrubber to stop, bd #_ (status __)
After final scrubbing, a single-bit error was found
After final scrubbing, a multi-bit error was found
```

The test ends with:

```
ECC      SCRUB: ECC Scrubbing..................... Running ---> FAILED
```

MC68040 Internal Cache (DCAC) Tests

This section describes the individual **DCAC** memory controller ASIC tests.

Entering **DCAC** without parameters causes all **DCAC** tests to execute in the order shown in the table below, except as noted.

To run an individual test, add that test name to the **DCAC** command. The individual tests are described in alphabetical order in the following pages.

Table 3-6. DCAC Test Group

Mnemonic	Description
DCAC_WT	Data Cache Writethrough
DCAC_CB	Data Cache Copyback
Executed only when specified:	
DCAC_RD	Display Cache Registers

Configuration of one parameter that the test uses may be accomplished through the use of the **CF** command:

```
162-Diag>CF DCAC
```

The prompt printed is:

```
DCAC Configuration Data:
```

This prompt allows you to set the base address of the **DCAC** configuration data block:

```
CF Structure Pointer =00004F88 ?
```

This prompt allows you to set the starting test address of the **DCAC** tests:

```
Test Address =0000E010 ?
```

This prompt allows you to set the size of cache memory to be tested:

```
Test Size =00000800 ?
```

Data Cache Copyback - DCAC_CB

Verifies the basic operation of the MC68040 Data Cache in Copyback mode. First a pattern is written to the test buffer and verified. The data cache is enabled in the copyback mode and the test buffer is then filled with the complement of the first pattern and verified (in cache as long as it fits). Then the cache is disabled without PUSHing the data to memory and the first pattern is verified in the test buffer. This operation is followed by a data cache PUSH operation and the second pattern is verified to be in the buffer.

Command Input:

`162-Diag>`**DCAC DCAC_CB**

Response/Messages:

After the command has been issued, the following line is printed:

```
DCAC DCAC_CB: Data Cache Copyback................ Running --->
```

If all parts of the test are completed correctly, then the test passes:

```
DCAC DCAC_CB: Data Cache Copyback................ Running ---> PASSED
```

If any part of the test fails, then the display appears as follows:

```
DCAC DCAC_CB: Data Cache Copyback................ Running ---> FAILED
(error message)
```

Here, (error message) is one of the following:

If compare errors are detected when the initial test pattern is verified:

```
Address =________, Expected =________, Actual =________
< up to 10 errors may be displayed here >
< Too many errors ... (if 10 errors detected) >

- Pattern 1 Memory Read Error
```

If compare errors are detected when the second test pattern is verified (should be in cache):

```
Address =________, Expected =________, Actual =________
< up to 10 errors may be displayed here >
< Too many errors ... (if 10 errors detected) >
```

```
Enabling Data Cache (copyback)- Pattern 2 Memory Read Error
```

If compare errors are detected when the initial test pattern is verified (in memory) after writing the second pattern (to the cache) and disabling the cache:

```
Address =________, Expected =________, Actual =________
< up to 10 errors may be displayed here >
 Too many errors ... (if 10 errors detected) >
```

```
Enabling Data Cache (copyback)Disabling Data Cache . . .
- Pattern 1 Memory Read Error
```

If compare errors are detected when the second test pattern is verified (should be in cache):

```
Address =________, Expected =________, Actual =________
< up to 10 errors may be displayed here >
< Too many errors ... (if 10 errors detected) >
```

```
Enabling Data Cache (copyback)Disabling Data Cache . . .
- Pattern 2 Cache Push Memory Read Error
```

Data Cache Registers - DCAC_RD

Displays the MC68040 registers that are used for cache control during the cache tests. Registers are read when the command is invoked. The displayed registers include the MC68040 Cache Control register, User Root Pointer register, Supervisor Root Pointer register, Instruction Transparent Translation registers 0 and 1, Data Transparent Translation registers 0 and 1, Translation Control register, and the MMU Status register.

Command Input:

```
162-Diag>DCAC DCAC_RD
```

Response/Messages:

After the command has been issued, the following lines are printed:

```
CACR =00000000
URP =00000000 SRP =00000000
ITT0 =00000000 ITT1 =00000000
DTT0 =00000000 DTT1 =E01FC040
TC =00000000 MMUSR=00000000

162-Diag>
```

Data Cache Writethrough - DCAC_WT

Verifies the operation of the MC68040 Data Cache in Writethrough mode. A pattern is written to the test buffer with the cache enabled. The data should not be cached, but should be written to the test buffer. The buffer is then verified which should cache the test data (pattern 1) and the cache is then disabled. The pattern is verified to be in the test buffer in memory. A complement pattern (pattern 2) is then written to the test buffer and verified. The cache is re-enabled and the test buffer data is expected to be the original test pattern.

Command Input:

```
162-Diag>DCAC DCAC_WT
```

Response/Messages:

After the command has been issued, the following line is printed:

```
DCAC DCAC_WT: Data Cache Writethrough............ Running --->
```

If all parts of the test are completed correctly, then the test passes:

```
DCAC DCAC_WT: Data Cache Writethrough............ Running ---> PASSED
```

If any part of the test fails, then the display appears as follows:

```
DCAC DCAC_WT: Data Cache Writethrough............ Running ---> FAILED
(error message)
```

Here, (error message) is one of the following:

If the test encounters data verify error on data pattern 1, after enabling data cache:

```
Test Buffer Addr. - $________
Enabling Data Cache (write-through) . . .
Address =________, Expected =________, Actual =________
< up to 10 errors may be displayed here >
< Too many errors ... (if 10 errors detected) >

- Pattern 1 Data Cache Read Error
```

If the test encounters data verify error on data pattern 1, after enabling and disabling data cache:

```
Test Buffer Addr.- $________
Enabling Data Cache (write-through)...
Disabling Data Cache . . .
Address =________, Expected =________, Actual =________
< up to 10 errors may be displayed here >
< Too many errors ... (if 10 errors detected) >

- Pattern 1 Memory Read Error
```

If the test encounters data verify error on data pattern 2, after enabling and disabling data cache:

```
Test Buffer Addr.-$________
Enabling Data Cache (write-through)...
Disabling Data Cache . . .
Address =________, Expected =________, Actual =________
< up to 10 errors may be displayed here >
< Too many errors ... (if 10 errors detected) >

- Pattern 2 Memory Read Error
```

If the test encounters data verify error on data pattern 1, after enabling, disabling, and re-enabling data cache:

```
Test Buffer Addr. - $________
Enabling Data Cache (write-through) . . .
Disabling Data Cache . . .
Enabling Data Cache (write-through) . . .
Address =________, Expected =________, Actual =________
< up to 10 errors may be displayed here >
< Too many errors ... (if 10 errors detected) >

- Pattern 1 Data Cache Disturb Read Error
```

Memory Management Unit (MMU) Tests

This chapter describes tests to verify basic functionality of the MC68040 internal Memory Management Unit (MMU). This includes basic register tests as well as functional tests involving building and using the MMU mapping tables required by the MC68040.

These tests check the interaction between the RAM used for testing and the MC68040 during master cycles performed by the CPU during tablewalk cycles and read-modify-write cycles to update status bits in the page entries. They also verify proper interpretation of certain status bits in the page table entries by the CPU.

User parameters are provided to allow for testing using either 4K or 8K page size with the default tables built at a specified address. Additionally, a debug tool is available to display an entire translation tree including descriptor addresses and their values for the specified table search logical address, root pointer, and page size.

Entering **MMU** without parameters causes all **MMU** tests, except as noted otherwise, to execute in the order shown in the table below.

To run an individual test, add that test name to the **MMU** command. The individual tests are described in alphabetical order on the following pages.

Table 3-7. MMU Test Group

Mnemonic	Description
TC	TC Register
RP	RP Register
WALK	Tablewalk Mapped Pages
MAPROM	Mapped ROM Read
USERPAGE	Used Page
MODPAGE	Modified Page
INVPAGE	Invalid Page
WPPAGE	Write Protect Page
Executed only when specified:	
DISPSRCH	Display Table Search
TBLBLD	Build Default Tables
TBLVERF	Verify Default Tables

You can use the **CF** command to configure the parameters that these tests use:

```
162-Diag>CF MMU
```

The command displays the default modifiable parameters:

```
MMU Configuration Data:
CF Structure Pointer      =00004FC0 ?
Page Size                 =00000000 ?
Table Memory              =0000E200 ?
Table Search Address      =FF800000 ?
Table Search Root Pointer =0000E200 ?
162-Diag>
```

The default setup of these parameters forces tables to be built in local memory, wherever it may be mapped.

An explanation of the parameters is as follows:

CF Structure Pointer
This parameter points to a structure in memory used to debug the test.

Page Size
This parameter selects the size of the physical pages of memory used by by the tests and utilities for all MMU translations. It corresponds to the Page Size bit in the MC68040 Translation Control Register. Default value = 0 (4KB page size); (0=4KB,1=8KB).

Table Memory
This parameter indicates where the MMU translation table will be built for all tests and utilities (if they are built at all). The appropriate root pointer will normally point to this address.

Table Search Address
This parameter is used by the Display Table Search utility as the Logical Address to be translated via a verbose table search operation.

Table Search Root Pointer
This parameter is used by the Display Table Search utility as the Root Pointer for the translation of the Table Search Address.

Display Table Search - DISPSRCH

Performs a verbose table search operation on the logical address contained in the configuration parameter "Table Search Address". It displays all descriptor pointers and the contents of each descriptor used for the logical-to-physical address translation. Normal as well as indirect page descriptors may be evaluated. The utility uses the configuration parameters "Table Search Root Pointer" and "Page Size" for the table search operation.

This utility can be used in a systems debug operation to verify MMU tables. The root pointer and page size of the tables in question along with the desired logical address should be entered using the **CF MMU** command before executing this command.

If any descriptor in the search is found to be invalid, a warning message will be displayed but the operation will still be completed.

Command Input:

Select the appropriate root pointer register (URP/SRP) from the register display and enter it along with the desired logical address using the **CF MMU** command.

Then enter the Display Table Search command to display the translation path and data information.

```
162-Diag>MMU DISPSRCH
Logical Address           =FF800000
Page Size                 =00001000
Root Pointer              =0000E200 (Domain Table pointer)
Domain Table[7F] Address  =0000E3FC
Domain Table[7F]          =0000E600 (Segment Table pointer)
Segment Table[60] Address =0000E780
Segment Table[60]         =0000E800 (Page Table pointer)
Page Table[00] Address    =0000E800
Page Table[00]            =FF80066D (Page Descriptor)
Physical Address          =FF800000
162-Diag>
```

To display information for other logical addresses, enter the logical address using the **CF MMU** command and re-execute the **MMU DISPSRCH** command.

Build Default Tables - TBLBLD

Builds the default translation tables used by the **MMU** tests. It is mainly intended for use in debug and development.

Tables are built starting at the address in the configuration parameters in Table Memory Start. All local resources are mapped one to one, and virtual spaces are defined for local RAM, ROM, and I/O. A VMEbus RAM module in the factory test system is also mapped (one to one). Supervisor Tables are built to decode all resources, and User Tables map only the VMEbus RAM module and a virtual address area in local RAM. All resources are marked non-cacheable and the I/O segments are also marked as serialized. The ROM segment is marked as write-protected. Command Input:

```
162-Diag>MMU TBLBLD
```

No error checking is performed during the table build operations other than inherent bus error reporting if an incorrect address has been entered for the Table Build Address parameter or if there is a problem with the memory where the tables are being built.

Verify Default Tables - TBLVERF

Verifies the default translation tables used by the **MMU** tests. It is mainly intended for use in debug and development.

Tables are verified for all resources mapped with the Build Default Tables command. This is accomplished by forcing a tablewalk on the first logical address of every page that has been mapped using the MC68040 PTEST instruction and reading the MMU Status Register to verify that the page is resident and that the " mapped" address matches the expected physical address. Error messages are displayed to indicate any detected discrepancies.

Both Instruction and Data MMUs are verified to properly translate logical addresses as expected. The MMUs are turned on for these verifications in order to force the tablewalks to occur.

Command Input:

`162-Diag>`**MMU TBLVERF**

Response/Messages:

If an error is discovered during the table verify, either of the following two messages may be displayed:

```
- Log.($XXXXXXXX) to Phys.($XXXXXXXX) not mapping ! -
- Page_Addr($XXXXXXXX) - MMU SR($XXXXXXXX) -
```

or

```
- Unexpected Bus Error !!! - at address $XXXXXXXX
```

TC Register Test - TC

Verifies that the page size bit in the Translation Control Register of the MC68040 can be set for both 4K and 8K pages.

Command Input:

```
162-Diag>MMU TC
```

Response/Messages:

After entering this command, the display should read as follows:

```
MMU     TC: TC Register............................. Running --->
```

If the test passes, the running message will be overwritten by the following message:

```
MMU     TC: TC Register............................. Running ---> PASSED
```

If a read of the register does not match the expected value, the test will display the expected and read values as follows:

```
MMU      TC: TC Register............................. Running ---> FAILED
TC Expected $4000 Read $0000
```

RP Register Test - RP

Performs a walking bit test (with complement) on both MC68040 Root Pointer registers (supervisor and user).

Command Input:

`162-Diag>`**MMU RP**

Response/Messages:

After entering this command, the display should read as follows:

```
MMU     RP: RP Register............................ Running --->
```

If the test passes, the running message will be overwritten by the following message:

```
MMU     RP: RP Register............................ Running ---> PASSED
```

If a read of the register does not match the expected value, the test will display the expected and read values as follows:

```
MMU      RP: RP Register............................ Running ---> FAILED
Suprv Root Pointer Reg.
Expected $00400000 Read $00000000
```

or:

```
MMU      RP: RP Register............................ Running ---> FAILED
User Root Pointer Reg.
Expected $00400000 Read $00000000
```

Tablewalk Mapped Pages - WALK

Builds and verifies the MMU translation tables for all local and VMEbus RAM spaces used by subsequent MMU tests. These two operations are described in the previous manual segments *Build Default Tables* and *Verify Default Tables*.

Command Input:

```
162-Diag>MMU WALK
```

Response/Messages:

After entering this command, the display should read as follows:

```
MMU    WALK: Table Walk............................ Running --->
```

If the test passes, the following message sequence will be displayed:

```
MMU    WALK: Table Walk............................ Running ---> PASSED
```

See the *Verify Default Tables* section for error message summary.

Mapped ROM Read Test - MAPROM

Verifies that the MMU can dynamically translate virtual addresses to physical addresses to read the onboard EPROM data. The ROM is read at its untranslated address as well as the virtual address it is mapped to and the data is expected to be the same. The entire ROM space data is verified in this way.

Command Input:

`167-diag>`**MMU MAPROM**

Response/Messages: After entering this command, the display should read as follows:

```
MMU     MAPROM: Mapped ROM Read..................... Running --->
```

If the test passes, the running message will be overwritten by the following message:

```
MMU     MAPROM: Mapped ROM Read..................... Running ---> PASSED
```

If a read of the pattern does not match the expected value, the following message type will be displayed:

```
MMU      MAPROM: Mapped ROM Read..................... Running ---> FAILED
Fnc = 5 Addr = $FF807000 Expect = $XXXXXXXX Read = $XXXXXXXX
TC   = 8000
RP   = 8400
Dom. Descr. $85FC = $8800
Seg. Descr. $8980 = $8A00
Page Descr. $8A1C = $FF80766D
```

Used Page Test - USEDPAGE

Verifies that the USED bits in the table and page descriptors are set by the MC68040 when an access forces a table search to be performed.

The USED bits of all three levels of descriptors for a particular logical address are cleared and the ATC is flushed. This is followed by a write access to the logical address under test which should force a table search and cause the bits to be set.

All three levels of descriptors are checked after each table search to ensure that the bit is set in each appropriate descriptor.

Command Input:

```
162-Diag>MMU USEDPAGE
```

Response/Messages:

After entering this command, the display should read as follows:

```
MMU     USEDPAGE: Used Page........................ Running --->
```

If the test passes, the running message will be overwritten by the following message:

```
MMU     USEDPAGE: Used Page........................ Running ---> PASSED
```

If all of the expected conditions are not present after the write access, the following message type or a subset of it will be displayed. The appropriate logical address and its descriptor addresses will always be indicated.

```
MMU     USEDPAGE: Used Page........................ Running ---> FAILED
- Recv'd X bus errors
- Used bit in domain descr. not set
- Used bit in segment descr. not set
- Used bit in page descr. not set
- Mod. bit in page descr. not set
Test address = XXXXXXXX
Domain, Segment, Page = XXXXXXXX, XXXXXXXX, XXXXXXXX
```

Modified Page Test - MODPAGE

Verifies that the MODIFIED bit in the page descriptor for a logical address is set by the MC68040 when an write access forces a table search to be performed. The MODIFIED bit of the page descriptor for a particular logical address is cleared and the ATC is flushed. This is followed by a write access to the logical address under test which should force a table search and cause the bit to be set. The appropriate page descriptor is checked after each access to ensure that the bit has been set.

Command Input:

`162-Diag>`**MMU MODPAGE**

Response/Messages:

After entering this command, the display should read as follows:

```
MMU     MODPAGE: Modified Page...................... Running --->
```

If the test passes, the running message will be overwritten by the following message:

```
MMU     MODPAGE: Modified Page...................... Running ---> PASSED
```

If all of the expected conditions are not present after the write access, the following message type or a subset of it will be displayed. The appropriate logical address and its descriptor addresses will always be indicated.

```
MMU      MODPAGE: Modified Page...................... Running ---> FAILED
- Recv'd X bus errors
- Used bit in domain descr. not set
- Used bit in segment descr. not set
- Used bit in page descr. not set
- Mod. bit in page descr. not set
Test address = XXXXXXXX
Domain, Segment, Page = XXXXXXXX, XXXXXXXX, XXXXXXXX
```

Invalid Page Test - INVPAGE

Verifies that if the segment or page descriptor for a logical address is marked invalid, and an access is attempted to that address, an access fault will be generated by the MC68040.

For each appropriate descriptor level and for each longword in the tested space, the descriptor type is set to the INVALID encoding and the ATC is flushed. The logical address under test is then accessed with the MMU enabled. This access is expected to cause an access fault (bus error).

Command Input:

`Diag-167>`**MMU INVPAGE**

Response/Messages:

After entering this command, the display should read as follows:

```
MMU     INVPAGE: Invalid Page....................... Running --->
```

If the test passes, the following message sequence will be displayed:

```
MMU     INVPAGE: Invalid Page....................... Running ---> PASSED
```

If the test access does NOT cause an access error to be detected, the following message type will be displayed:

```
MMU      INVPAGE: Invalid Page....................... Running ---> FAILED
Invalid Segment Descriptor
- Bus Error NOT Detected !!
```

If the test access causes multiple access errors to be detected, the following message type will be displayed:

```
MMU      INVPAGE: Invalid Page....................... Running ---> FAILED
Invalid Segment Descriptor
- Recv'd x bus errors
```

Write Protect Page Test - WPPAGE

Verifies that if the page descriptor for a logical address is marked write-protected, and a write access is attempted to that address, an access fault will be generated by the MC68040.

For each longword in the tested space, the page descriptor write access attribute is set to the WRITE-PROTECTED encoding and the ATC is flushed. A write access to the logical address under test is then attempted with the MMU enabled.

This access is expected to cause an access fault (bus error).

Command Input:

`167-DIAG>`**MMU WPPAGE**

Response/Messages:

After entering this command, the display should read as follows:

```
MMU     WPPAGE: Write Protect Page.................. Running --->
```

If the test passes, the following message sequence will be displayed:

```
MMU     WPPAGE: Write Protect Page.................. Running ---> PASSED
```

If the test access does NOT cause an access error to be detected, the following message type will be displayed:

```
MMU      WPPAGE: Write Protect Page.................. Running ---> FAILED
Write-Protected Segment Descriptor
- Bus Error NOT Detected
```

If the test access causes multiple access errors to be detected, the following message type will be displayed:

```
MMU      WPPAGE: Write Protect Page.................. Running ---> FAILED
Write-Protected Segment Descriptor
- Recv'd x bus errors
```

VME Interface Chip (VME2) Tests

These sections describe the individual VME2 tests.

Entering **VME2** without parameters causes all **VME2** tests to execute in the order shown in the table below.

To run an individual test, add that test name to the **VME2** command. The individual tests are described in alphabetical order on the following pages.

Table 3-8. VME2 Test Group

Mnemonic	Description
REGA	Register Access
REGB	Register Walking Bit
TMRA	Tick Timer 1 Increment
TMRB	Tick Timer 2 Increment
TMRC	Prescaler Clock Adjust
TMRD	Tick Timer 1 No Clear On Compare
TMRE	Tick Timer 2 No Clear On Compare
TMRF	Tick Timer 1 Clear On Compare
TMRG	Tick Timer 2 Clear On Compare
TMRH	Tick Timer 1 Overflow Counter
TMRI	Tick Timer 2 Overflow Counter
TMRJ	Watchdog Timer Counter
TMRK	Watchdog Timer Board Fail
TACU	Timer Accuracy
SWIA	Software Interrupts (Polled Mode)
SWIB	Software Interrupts (Processor Interrupt Mode)
SWIC	Software Interrupts Priority

Register Access - REGA

Verifies that the registers at offsets 0 through 84 can be read accessed. The read access algorithm is performed using eight, sixteen, and thirty-two bit data sizes.

Command Input:

```
162-Diag>VME2 REGA
```

Response/Messages:

After the command has been issued, the following line is printed:

```
VME2    REGA: Register Access....................... Running --->
```

If all parts of the test are completed correctly, then the test passes:

```
VME2    REGA: Register Access....................... Running ---> PASSED
```

If any part of the test fails, then the display appears as follows:

```
VME2    REGA: Register Access....................... Running ---> FAILED
```

```
VME2/REGA Test Failure Data:

Unsolicited Exception:
   Exception Time PC/IP _____
                Vector _
Access Fault Information:
               Address ________
                  Data ________
           Access Size _
           Access Type _
    Address Space Code __
 reg_a:
            Data Width __ bits
```

NOTES:

1. All data is displayed as hexadecimal values.
2. The Access Fault Information is only displayed if the exception was an Access Fault (Bus Error).
3. Access size is displayed in bytes.
4. Access type is 0 or 1 for write or read, respectively.
5. The address space code message uses the following codes: 1, 2, 5, 6, and 7 for user data, user program, supervisor data, supervisor program, and MPU space, respectively. All address space codes listed above may not be applicable to any single microprocessor type.

Register Walking Bit - REGB

Verifies that certain bits in the VMEchip2 ASIC user registers can be set independently of other bits in the VMEchip2 ASIC user registers. This test also assures that the VMEchip2 ASIC user registers can be written without a Data Fault (Bus Error). The VMEchip2 register walking bit test is implemented by first saving the initial state of the Local Control and Status Registers (LCSR). All eligible bits are then initialized to zero. This initialization is verified. A one is walked through the LCSR bit array and the entire register bit field is verified after each write. All eligible bits are then initialized to one. This initialization is then verified. A zero is walked through the LCSR bit array and the entire register bit field is verified after each write. The initial state of the LCSR is restored except for the LCSR Prescaler Counter register.

Command Input:

```
162-Diag>VME2 REGB
```

Response/Messages:

After the command has been issued, the following line is printed:

```
VME2    REGB: Register Walking Bit.................. Running --->
```

If all parts of the test are completed correctly, then the test passes:

```
VME2    REGB: Register Walking Bit.................. Running ---> PASSED
```

If any part of the test fails, then the display appears as follows:

```
VME2     REGB: Register Walking Bit.................. Running ---> FAILED
(error message)
```

Here, (error message) is one of the following:

If a bit in the LCSR cannot be initialized:

```
bfverf: Bit Field Initialization Error.
                Address ________
              Read Data ________
     Failing Bit Number __ (&__)
     Expected Bit Value _
       Actual Bit Value _
       Exempt Bits Mask ________
```

If a bit in the LCSR fails to respond properly to the walking bit algorithm:

```
regvrf: bit error:
                Address ________
              Read Data ________
     Failing Bit Number __ (&__)
     Expected Bit Value __
       Actual Bit Value __
       Exempt Bits Mask ________

       Written Register ________
     Written Bit Number __ (&__)
           Written Data __
```

If an unexpected interrupt is received while executing the test:

```
VME2/REGB Test Failure Data:

Unsolicited Exception:
  Exception Time PC/IP _____
                Vector _
Access Fault Information:
               Address ________
                  Data ________
           Access Size _
           Access Type _
    Address Space Code __
```

Software Interrupts (Polled Mode) - SWIA

Verifies that all software interrupts (1 through 7) can be generated and that the appropriate status is set.

Command Input:

```
162-Diag>VME2 SWIA
```

Response/Messages:

After the command has been issued, the following line is printed:

```
VME2    SWIA: Software Interrupts Polled............ Running --->
```

If all parts of the test are completed correctly, then the test passes:

```
VME2    SWIA: Software Interrupts Polled............ Running ---> PASSED
```

If any part of the test fails, then the display appears as follows:

```
VME2     SWIA: Software Interrupts Polled............ Running ---> FAILED
(error message)
```

Here, (error message) is one of the following:

The VMEchip2 local bus interrupter enable register is cleared and the local bus interrupter status register is read to verify that no interrupt status bits are set. If any bits are set:

```
Interrupt Status Register is not initially cleared
Status: Expected =00000000, Actual =________
```

Prior to asserting any SWI set bit, and with local bus interrupter enable register SWI bits asserted, the local bus interrupter status register is again checked to verify that no status bits became true:

```
Interrupt Status Register is not clear
Status: Expected =________, Actual =________
State : IRQ Level =__, SWI__, VBR =__
```

As the different combinations of SWI, interrupt level, and, interrupt vector are asserted, verification is made that the expected SWI interrupt status bit did become true, and only that status bit became true, or else the following message appears:

```
Unexpected status set in Interrupt Status Register
Status: Expected =________, Actual =________
State : IRQ Level =__, SWI__, VBR =__
```

After the interrupt is generated, the clear bit for the current SWI interrupter is asserted and a check is made to verify the status bit cleared:

```
Interrupt Status Bit did not clear
Status: Expected =________, Actual =________
State : IRQ Level =__, SWI__, VBR =__
```

Software Interrupts (Processor Interrupt Mode) - SWIB

Verifies that all software interrupts (levels 1 through 7) can be generated and received and that the appropriate status is set.

Command Input:

```
162-Diag>VME2 SWIB
```

Response/Messages:

After the command has been issued, the following line is printed:

```
VME2    SWIB: Software Interrupts Interrupt......... Running --->
```

If all parts of the test are completed correctly, then the test passes:

```
VME2    SWIB: Software Interrupts Interrupt......... Running ---> PASSED
```

If any part of the test fails, then the display appears as follows:

```
VME2     SWIB: Software Interrupts Interrupt......... Running ---> FAILED
(error message)
```

Here, (error message) is one of the following:

The interrupt enable register is cleared and status bits are read to verify that none are true:

```
Interrupt Status Register is not initially cleared
Status: Expected =________, Actual =________
```

Prior to asserting any SWI set bit, and with local bus interrupter enable register SWI bits asserted, the local bus interrupter status register is checked to verify that no status bit became true:

```
Interrupt Status Register is not clear
Status: Expected =________, Actual =________
State : IRQ Level =__, SWI__, VBR =__
```

If the MPU is an M88000 family processor, the exception vector number is checked to make sure that the exception received was that of the interrupt (exception number 1):

```
Incorrect Vector type
Vector: Expected =____, Actual =____
Status: Expected =____, Actual =____
State : IRQ Level =___, SWI__, VBR =___
```

If the received interrupt vector is not that of the programmed interrupt vector:

```
Unexpected Vector taken
Vector: Expected =____, Actual =____
Status: Expected =____, Actual =____
State : IRQ Level =___, SWI__, VBR =___
```

If the received interrupt level is not that of the programmed interrupt level:

```
Incorrect Interrupt Level
Level : Expected =____, Actual =____
State : IRQ Level =____, SWI__, VBR =____
```

If the programmed interrupt did not occur:

```
Software Interrupt did not occur
Status: Expected =____, Actual =____
State : IRQ Level =___, SWI__, VBR =___
```

The VMEchip2 Interrupt Status Register is checked for the proper interrupt status bit to be active:

```
Unexpected status set in Interrupt Status Register
Status: Expected =____, Actual =____
State : IRQ Level =___, SWI__, VBR =___
```

If, after receiving an interrupt, the interrupt status cannot be negated by writing the interrupt clear register:

```
Interrupt Status Bit did not clear
Status: Expected =____, Actual =____
State : IRQ Level =___, SWI__, VBR =___
```

Software Interrupts Priority - SWIC

Verifies that all software interrupts (1 through 7) occur in the priority set by the hardware.

Command Input:

162-Diag>**VME2 SWIC**

Response/Messages:

After the command has been issued, the following line is printed:

```
VME2    SWIC: Software Interrupts Priority.......... Running --->
```

If all parts of the test are completed correctly, then the test passes:

```
VME2    SWIC: Software Interrupts Priority.......... Running ---> PASSED
```

If any part of the test fails, then the display appears as follows:

```
VME2     SWIC: Software Interrupts Priority.......... Running ---> FAILED
(error message)
```

Here, (error message) is one of the following:

The interrupt enable register is cleared and status bits are read to verify that none are true:

```
Interrupt Status Register is not initially cleared
Status: Expected =________, Actual =________
```

If the MPU is an M88000 family processor, the exception vector number is checked to make sure that the exception received was that of the interrupt (exception number 1):

```
Incorrect Vector type
Vector: Expected =__, Actual =__
Status: Expected =________, Actual =________
State : IRQ Level =___, SWI__, VBR =__
```

If the received interrupt vector is not that of the programmed interrupt vector:

```
Unexpected Vector taken
Vector: Expected =__, Actual =__
Status: Expected =________, Actual =________
State : IRQ Level =___, SWI__, VBR =__
```

If the received interrupt level is not that of the programmed interrupt level:

```
Incorrect Interrupt Level
Level : Expected =__, Actual =__
State : IRQ Level =___, SWI__, VBR =__
```

If the programmed interrupt did not occur:

```
Software Interrupt did not occur
Status: Expected =________, Actual =________
State : IRQ Level =___, SWI__, VBR =__
```

The VMEchip2 Interrupt Status Register is checked for the proper interrupt status bit to be active:

```
Unexpected status set in Interrupt Status Register
Status: Expected =________, Actual =________
State : IRQ Level =___, SWI__, VBR =__
```

If, after receiving an interrupt, the interrupt status cannot be negated by writing the interrupt clear register:

```
Interrupt Status Bit did not clear
Status: Expected =________, Actual =________
State : IRQ Level =___, SWI__, VBR =__
```

Timer Accuracy Test - TACU

Performs a four point verification of the VMEChip2 ASIC timer and prescaler circuitry using the on-board Real Time Clock (RTC) as a timing reference.

The RTC seconds register is read and the stop, write, and read bits are verified to be negated to ensure that the RTC is in the correct state for use by the firmware-based diagnostics.

The prescaler calibration register is checked to verify that it contains one of four legal MPU clock calibration values.

Both 32 bit tick timers are programmed to accumulate count, starting at zero, for a period of time determined by the RTC. The accumulated count is verified to be within a predetermined window.

The upper 24 bits of the prescaler counter register is read at two intervals whose timing is determined by the RTC. The difference count is verified to be within a predetermined window.

Command Input:

`162-Diag>`**VME2 TACU**

Response/Messages:

After the command has been issued, the following line is printed:

```
VME2     TACU: Timer Accuracy................... Running --->
```

If all parts of the test are completed correctly, then the test passes:

```
VME2     TACU: Timer Accuracy................... Running ---> PASSED
```

If any part of the test fails, then the display appears as follows:

```
VME2     TACU: Timer Accuracy................... Running ---> FAILED
(error message)
```

Here, (error message) is one of the following:

If the RTC is stopped:

```
RTC is stopped, invoke SET command.
```

If the RTC is in the write mode:

```
RTC is in write mode, invoke SET command.
```

If the RTC is in the read mode:

```
RTC is in read mode, invoke SET command.
```

If the prescaler calibration register does not contain one of four legal MPU clock calibration values:

```
Illegal prescaler calibration:
Expected EF, EC, E7, or DF, Actual =__
```

If tick timer accuracy is out of tolerance:

```
Timer counter register read (greater/less) than expected
Address =________, Expected =________, Actual =________
```

If prescaler counter register accuracy is out of tolerance, the prescaler counter address and expected and actual difference counts are displayed:

```
Prescaler delta was (greater/less) than expected
Address =________, Expected =________, Actual =________
```

If the RTC seconds register does not increment during the test:

```
RTC seconds register didn't increment
```

Tick Timer Increment - TMRA, TMRB

Verifies that Timer *x* Counter Register (*x* = 1 or 2) can be set to 0, and, that Timer *x* Counter Register value increments when enabled. The Timer is initialized by writing 0 to the Tick Timer Counter Register. The Clear On Compare mode is disabled by writing the COC*x* bit in the Tick Timer Control Register. The Timer is enabled by the EN*x* bit in the Tick Timer Control Register. The MPU executes a time delay loop, then disables Tick Timer *x*. The Tick Timer Control Register is read to see if it incremented from its initial value of 0. **TMRA** specifies Tick Timer 1. **TMRB** specifies Tick Timer 2.

Command Input:

`162-Diag>`**VME2 TMRA**

or:

`162-Diag>`**VME2 TMRB**

Response/Messages:

Note that in all responses shown below, the response `"TMRx: Timer n"` is `TMRA: Timer 1` or `TMRB: Timer 2`, depending upon which test set is being performed.

After the command has been issued, the following line is printed:

```
VME2    TMRx: Timer n Increment..................... Running --->
```

If all parts of the test are completed correctly, then the test passes:

```
VME2    TMRx: Timer n Increment..................... Running ---> PASSED
```

If any part of the test fails, then the display appears as follows:

```
VME2     TMRx: Timer n Increment..................... Running ---> FAILED
(error message)
```

Here, (error message) is one of the following:

```
Tick Timer _ Counter did not clear.
```

```
Tick Timer _ Counter did not increment.
```

Prescaler Clock Adjust - TMRC

Proves that the Prescaler Clock Adjust register can vary the period of the tick timer input clock. The test fails if the Prescaler Clock Adjust register has not been previously initialized to a nonzero value. Two MPU timing loops are executed, the first with a "low" Prescaler Clock Adjust register value, the second with a "high" value. Timer 1 of the VMEchip2 is used for reference in this test. The first MPU loop count is compared with the second MPU loop count. The first MPU loop count is expected to be smaller than the second. The Prescaler Clock Adjust register value is restored upon correct test execution.

Command Input:

```
162-Diag>VME2 TMRC
```

Response/Messages:

After the command has been issued, the following line is printed:

```
VME2    TMRC: Prescaler Clock Adjust................ Running --->
```

If all parts of the test are completed correctly, then the test passes:

```
VME2    TMRC: Prescaler Clock Adjust................ Running ---> PASSED
```

If any part of the test fails, then the display appears as follows:

```
VME2    TMRC: Prescaler Clock Adjust................ Running ---> FAILED
```

If Prescaler Clock Adjust register was = 0:

```
Prescaler Clock Adjust reg was not initialized
```

A non-incrementing timer gives the following for first loop timeouts:

```
Low value:Timed out waiting for compare (ITIC1) ________ to assert.
```

or, for last loop timeouts:

```
High value:Timed out waiting for compare (ITIC1) ________ to assert
```

If the Prescaler Clock Adjust did not vary tick period:

```
Prescaler Clock Adjust did not vary tick period.
Loop1=________, Loop2=________.
```

Tick Timer No Clear On Compare - TMRD, TMRE

Verifies the Tick Timers No Clear On Compare mode. The Timer is initialized by writing 0 to the Tick Timer Counter Register. The Clear On Compare mode is disabled by writing the COC*x* bit in the Tick Timer Control Register. The compare value is initialized by writing $55aa to the Tick Timer Compare Register. The Timer is enabled by the EN*x* bit in the Tick Timer Control Register. After starting the timer, the MPU enters a time delay loop while testing for Tick Timer compare. Tick Timer compare is sensed by reading the TIC*x* bit in the Local Bus Interrupter Status Register. The Timer is stopped when Timer Compare is sensed, or an MPU loop counter register decrements to 0 (timeout). If the MPU loop counter did not time out, the Timer Counter Register is read to make sure that it was not cleared on compare. **TMRD** specifies Tick Timer 1. **TMRE** specifies Tick Timer 2.

Command Input:

`162-Diag>`**VME2 TMRD**

or:

`162-Diag>`**VME2 TMRE**

Response/Messages:

Note that in all responses shown below, the response `"TMRx: Timer n"` is `TMRD: Timer 1` or `TMRE: Timer 2`, depending upon which test set is being performed.

After the command has been issued, the following line is printed:

```
VME2    TMRx: Timer n No Clear On Compare........... Running --->
```

If all parts of the test are completed correctly, then the test passes:

```
VME2    TMRx: Timer n No Clear On Compare........... Running ---> PASSED
```

If any part of the test fails, then the display appears as follows:

```
VME2     TMRx: Timer n No Clear On Compare........... Running ---> FAILED
(error message)
```

Here, (error message) is one of the following:

```
Tick Timer ___: Counter did not clear.

Timer Counter Register = ________/________ (address/data)

Tick Timer ____: Timed out waiting for compare (ITICn).

Tick Timer ____: Timer cleared on compare.
Timer Counter Register = ________/________ (address/data)
```

Tick Timer Clear On Compare - TMRF, TMRG

Verifies the Tick Timers Clear On Compare mode. The Timer is initialized by writing 0 to the Tick Timer Counter Register. The Clear On Compare mode is enabled by writing the COC*x* bit in the Tick Timer Control Register. The compare value is initialized by writing $55aa to the Tick Timer Compare Register. The Timer is enabled by the EN*x* bit in the Tick Timer Control Register. After starting the timer, the MPU enters a time delay loop while testing for Tick Timer compare. Tick Timer compare is sensed by reading the TIC*x* bit in the Local Bus Interrupter Status Register. The Timer is stopped when Timer Compare is sensed, or an MPU loop counter register decrements to 0 (timeout). If the MPU loop counter did not time out, the Timer Counter Register is read to make sure that it was cleared on compare. **TMRF** specifies Tick Timer 1. **TMRG** specifies Tick Timer 2.

Command Input:

`162-Diag>`**VME2 TMRF**

or:

`162-Diag>`**VME2 TMRG**

Response/Messages:

Note that in all responses shown below, the response "`TMRx: Timer` *n*" is `TMRF: Timer 1` or `TMRG: Timer 2`, depending upon which test set is being performed.

After the command has been issued, the following line is printed:

```
VME2    TMRx: Timer n Clear On Compare.............. Running --->
```

If all parts of the test are completed correctly, then the test passes:

```
VME2    TMRx: Timer n Clear On Compare.............. Running ---> PASSED
```

If any part of the test fails, then the display appears as follows:

```
VME2     TMRx: Timer n Clear On Compare.............. Running ---> FAILED
(error message)
```

Here, (error message) is one of the following:

```
Tick Timer ____: Counter did not clear.
Timer Counter Register = ________/________ (address/data)

Tick Timer ____: Timed out waiting for compare (ITIC____).

Tick Timer ____: Timer didn't clear on compare.
Timer Counter Register = ________/________ (address/data)
```

Overflow Counter - TMRH, TMRI

Verifies that the overflow counter is enabled and a count of timer overflow is expected to accumulate. The COVF bit in the timer control register is asserted and OVF bit is verified to be clear. The timer counter register is set to zero, the timer compare register is loaded with the value $55aa, and the timer is enabled. When TIC(1/2) becomes true, the timer is disabled and the timer overflow counter register is checked to see that the resultant overflow was counted. **TMRH** specifies Tick Timer 1 Overflow Counter. **TMRI** specifies Tick Timer 2 Overflow Counter.

Command Input:

`162-Diag>`**VME2 TMRH**

or:

`162-Diag>`**VME2 TMRI**

Response/Messages:

Note that in all responses shown below, the response "`TMR`*`x`*`: Timer `*`n`*" is `TMRH: Timer 1` or `TMRI: Timer 2`, depending upon which test set is being performed.

After the command has been issued, the following line is printed:

```
VME2    TMRx: Timer n Overflow  Counter............. Running --->
```

If all parts of the test are completed correctly, then the test passes:

```
VME2    TMRx: Timer n Overflow  Counter............. Running ---> PASSED
```

If any part of the test fails, then the display appears as follows:

```
VME2     TMRx: Timer n Overflow  Counter............. Running ---> FAILED
(error message)
```

Here, (error message) is one of the following:

```
Timer ____: Overflow Counter did not clear.
Timer Control Register = ________

Tick Timer ____: Counter did not clear.
Timer Counter Register = ________/________ (address/data)

Tick Timer ____: timeout waiting for ITIC____

Tick Timer ____: Overflow counter did not increment
Timer Control Register = ________
```

Watchdog Timer Counter - TMRJ

Verifies functionality at all programmable timing values. This test also checks watchdog timer clear status and timeout functions. The following is done for all programmable watchdog timeouts:

1. Check for linear timeout period with respect to previous timeout.
2. Verify that timeout status can be cleared.

Command Input:

```
162-Diag>VME2 TMRJ
```

Response/Messages:

After the command has been issued, the following line is printed:

```
VME2    TMRJ: Watchdog Timer Counter................ Running --->
```

If all parts of the test are completed correctly, then the test passes:

```
VME2    TMRJ: Watchdog Timer Counter................ Running ---> PASSED
```

If any part of the test fails, then the display appears as follows:

```
VME2     TMRJ: Watchdog Timer Counter................ Running ---> FAILED
(error message)
```

Here, (error message) is one of the following:

```
Watchdog failed to timeout: mloops=________

out of tolerance
          time out code ________
           actual loops ________
         expected loops ________
            lower limit ________
            upper limit ________

time out status (WDTO bit) could not be cleared
```

Watchdog Timer Board Fail - TMRK

Tests the watchdog timer in board fail mode by setting up a watchdog timeout and verifying the status of the VMEchip2 BRFLI status bit in the Board Control register. This test verifies BRFLI for WDBFE both negated and asserted states.

Command Input:

162-Diag>**VME2 TMRK**

Response/Messages:

After the command has been issued, the following line is printed:

```
VME2    TMRK: Watchdog Timer Board Fail............. Running --->
```

If all parts of the test are completed correctly, then the test passes:

```
VME2    TMRK: Watchdog Timer Board Fail............. Running ---> PASSED
```

If any part of the test fails, then the display appears as follows:

```
VME2     TMRK: Watchdog Timer Board Fail............. Running ---> FAILED
(error message)
```

Here, (error message) is one of the following:

```
Watchdog failed to timeout: wdbfe=________, mloops=________

BRFLI (at $________) was High, it should have been Low

BRFLI (at $________) was Low, it should have been High

wdog: time out status (WDTO bit) could not be cleared
```

LAN Coprocessor for Ethernet (LANC) Tests

This section describes the individual Local Area Network Coprocessor (i82596) for Ethernet (**LANC**) tests. The terms **LANC** and 82596 are used interchangeably in the following **LANC** test group explanation text.

The 82596, as an intelligent, high-performance LAN coprocessor, executes high-level commands, command chaining, and interprocessor communications via shared memory. This relieves the host CPU of many tasks associated with network control; all time-critical functions are performed independently of the CPU, which greatly improves network performance.

The 82596 manages all IEEE 802.3 Medium Access Control and channel interface functions, for example, framing, preamble generation and stripping, source address insertion, destination address checking, short frame detection, and automatic length-field handling. The 82596 supports serial data rates up to 20Mb/s.

Entering **LANC** without parameters causes all **LANC** tests to execute in the order shown in the table below, except as noted.

To run an individual test, add that test name to the **LANC** command.

Table 3-9. LANC Test Group

Mnemonic	Description
CST	Chip Self Test
BERR	Bus Error
IRQ	Interrupt Request
DUMP	Dump Configuration/Registers
DIAG	Diagnose Internal Hardware
ILB	Internal Loopback
ELBT	External Loopback Transceiver
Executed only when specified:	
ELBC	External Loopback Cable
MON	Monitor (Incoming Frames) Mode
TDR	Time Domain Reflectometry

The individual tests are described in alphabetical order on the following pages.

The error message displays following the explanation of an individual **LANC** test pertain to the test being discussed. See also the subsection *Additional Error Messages* at the end of this section for messages that apply to all tests in the group.

You can use the **CF LANC** command to configure the transmit to receive loop count (32) for the **ELBC**, **ELBT**, and **ILB** tests:

```
162-Diag>CF LANC
```

Chip Self Test - CST

Verifies that the 82596 self-test mode (command) can be executed, and also verifies that the self-test results (expected results) match the actual results. The 82596 provides the results of the self-test at the address specified by the self-test PORT command. The self-test command checks the following blocks (of the 82596):

❏ ROM
The contents of the entire ROM is sequentially read into a Linear Feedback Shift Register (LFSR). The LFSR compresses the data and produces a signature unique to one set of data. The results of the LFSR are then compared to a known good ROM signature. The pass or fail result and the LFSR contents are written into the address specified by the self-test PORT command.

❏ Parallel Registers
The micro machine performs write and read operations to all internal parallel registers and checks the contents for proper values. The pass or fail result is then written into the address specified by the self-test PORT command.

❏ Bus Throttle Timers
The micro machine performs an extensive test of the Bus Throttle timer cells and decrementation logic. The counters are enabled and the contents are checked for proper values. The pass or fail result is then written to the address specified by the self-test PORT command.

❏ Diagnose
The micro machine issues an internal diagnose command to the serial subsystem. The pass or fail result of the Diagnose command is then written into the address specified by the self-test PORT command.

Command Input:

```
162-Diag>LANC CST
```

Response/Messages:

After the command has been issued, the following line is printed:

```
LANC    CST: Chip Self Test........................ Running --->
```

If all parts of the test are completed correctly, then the test passes:

```
LANC    CST: Chip Self Test........................ Running ---> PASSED
```

If any failures occur, the following is displayed (more descriptive text then follows):

```
LANC     CST: Chip Self Test.......................... Running ---> FAILED
(error message)
```

Here, (error message) is one of the following:

If the expected results do not match (equal) the actual results of the 82596 self-test command results:

```
LANC Chip Self-Test Error: Expected =00000000, Actual =10040000
```

Diagnose Internal Hardware - DIAG

Verifies that the Diagnose command of the 82596 can be executed, and that an error-free completion status is returned. The Diagnose command triggers an internal self-test procedure that checks the 82596 hardware, which includes the following:

- ❏ Exponential Backoff Random Number Generator (Linear Feedback Shift Register).
- ❏ Exponential Backoff Timeout Counter
- ❏ Slot Time Period Counter
- ❏ Collision Number Counter
- ❏ Exponential Backoff Shift Register
- ❏ Exponential Backoff Mask Logic
- ❏ Timer Trigger Logic

The Channel Interface Module of the 82596 performs the self-test procedure in two phases: Phase 1 tests the counters and Phase 2 tests the trigger logic.

During Phase 1, the Linear Feedback Shift Register (LFSR) and the Exponential Backoff Timer, Slot Timer, and Collision Counters are checked.

Phase 1:

1. Simultaneously resets all counters and shift registers.
2. Starts counting and shifting the registers until the Exponential Backoff Shift Register reaches all ones.
3. Checks the Exponential Backoff Shift Register for all ones when the LFSR content is all ones in its least significant bits.
4. Stops counting when the LFSR (30 bits) reaches a specific state, and Exponential Backoff Counter (10 bits) wraps from "All Ones" to "All Zeros". Simultaneously, the Slot Time counter switches from 01111111111 to 10000000000, and the collision counter (4 bits) wraps from "All Ones" to "All Zeros".
5. Phase 1 is successful if the 10 least significant bits (when applicable) of all four counters are "All Zeros".

Phase 2:

1. Resets Exponential Backoff Shift Register and all counters.
2. Temporarily configures Exponential Backoff logic internally, as listed:

SLOT-TIME	= $3
LIN-PRIO	= $6
EXP-PRIO	= $3
BOF-MET	= $0

3. Emulates transmission and collision, internally.
4. Returns a "Passed" status if the most significant bit of Exponential Backoff Shift Register is 1
   or
   returns a "Failed" status if the most significant bit of Exponential Backoff Shift Register is 0 and repeats Step 3.

Command Input:

```
162-Diag>LANC DIAG
```

Response/Messages:

After the command has been issued, the following line is printed:

```
LANC    DIAG: Diagnose Internal Hardware........... Running --->
```

If all parts of the test are completed correctly, then the test passes:

```
LANC    DIAG: Diagnose Internal Hardware........... Running ---> PASSED
```

If any failures occur, the following is displayed (more descriptive text then follows):

```
LANC     DIAG: Diagnose Internal Hardware........... Running ---> FAILED
(error message)
```

This failure results if the Diagnose command fails:

```
DIAGNOSE Command Completion Status Error:
OK-Bit =0, F(ail)-Bit =1
```

Dump Configuration/Registers - DUMP

Verifies that the Dump command of the 82596 can be executed, and that an error free completion status is returned. The Dump command instructs the 82596 to transfer the configuration parameters and contents of other registers from the Channel Interface Module via RCV-FIFO by Receive Unit to memory.

The test issues the Dump command to the 82596 and waits for two seconds. Once the delay has expired, the test verifies the command completion status. The 82596 performs the following sequence upon the receipt of the Dump command:

- Starts Action command.
- Writes Dump command byte to TX-FIFO.
- Waits for completion of DUMP.
- Prepares STATUS word with C=1, B=0, and OK=1.
- Completes Action command.

Command Input:

`162-Diag>`**LANC DUMP**

Response/Messages:

After the command has been issued, the following line is printed:

```
LANC    DUMP: Dump Configuration/Registers.......... Running --->
```

If all parts of the test are completed correctly, then the test passes:

```
LANC    DUMP: Dump Configuration/Registers.......... Running ---> PASSED
```

If any failures occur, the following is displayed (more descriptive text then follows):

```
LANC     DUMP: Dump Configuration/Registers.......... Running ---> FAILED
(error message)
```

This failure is the result of the Dump command failing:

```
Dump Status Error: Expected =A006, Actual =8006
```

External Loopback Cable - ELBC

Verifies that the 82596 can be operated in the "External Loopback with the LPBK pin not activated" mode.

The 82596 has three modes of loopback: Internal Loopback, External Loopback with the LPBK pin activated, and External Loopback with the LPBK pin not activated. The LPBK pin is connected to the accompanying Ethernet Serial Interface (ESI - 82C501AD) chip. The ESI is then connected to the pulse transformer (PE64102), which in turn is connected to the Ethernet Connector.

In External Loopback mode the 82596 transmits and receives simultaneously at a full rate. This allows checking external hardware as well as the serial link to the transceiver interface. The LPBK pin is used to inform the external hardware (ESI) of the establishment of a transmit to receive connection.

The test sets up a data packet (incrementing data pattern) to be transmitted, transmits it, and waits for the reception of the data. Once the data is received, the data is verified to the data transmitted. This transmit to receive loop is performed 32 times. Use the **CF LANC** command to change the loop count.

Note that this test does not execute when the **LANC** test group is executed (**LANC** with no arguments). This test is supplied only for diagnostic purposes. It requires a properly set up Ethernet network (cable).

Command Input:

```
162-Diag>LANC ELBC
```

Response/Messages:

After the command has been issued, the following line is printed:

```
LANC    ELBC: External Loopback Cable............... Running --->
```

If all parts of the test are completed correctly, then the test passes:

```
LANC    ELBC: External Loopback Cable............... Running ---> PASSED
```

If any failures occur, the following is displayed (more descriptive text then follows):

```
LANC    ELBC: External Loopback Cable............... Running ---> FAILED
(error message)
```

Once the data packet has been set up to be transmitted, the test instructs the 82596 (through the Command Unit) to transmit the data packet. This failure is the result of the 82596 completing with a transmit data error. The status bits of the error message display indicate the source of the problem:

```
TRANSMIT Command Completion Status Error:
OK-Bit =0, ABORT-Bit =0, STATUS-Bits =0010
```

STATUS-Bits breakdown:

Bit #6 Late collision. A late collision (a collision after the slot time elapsed) is detected.

Bit #5 No Carrier Sense signal during transmission. Carrier Sense signal is monitored from the end of Preamble transmission until the end of the Frame Check Sequence for TONOCRS = 1 (Transmit On No Carrier Sense Mode); it indicates that transmission has been executed despite a lack of CRS. For TONOCRS = 0 (Ethernet mode), this bit also indicates unsuccessful transmission (transmission stopped when lack of Carrier Sense has been detected).

Bit #4 Transmission unsuccessful (stopped) due to Loss of Clear to Send signal.

Bit #3 Transmission unsuccessful (stopped) due to DMA Underrun; i.e., the system did not supply data for transmission.

Bit #2 Transmission Deferred, i.e., transmission was not immediate due to previous link activity.

Bit #1 Heartbeat Indicator. Indicates that after a previously performed transmission, and before the most recently performed transmission, (Interframe Spacing) the CDT signal was monitored as active. This indicates that the Ethernet Transceiver Collision Detect logic is performing well. The Heartbeat is monitored during Interframe Spacing period.

Bit #0 Transmission attempt was stopped because the number of collisions exceeded the maximum allowable number of retries.

Once the data packet is transmitted successfully, the test waits for four seconds for the receipt of the data. This failure is the result of the timeout (four seconds) expiring:

```
RECEIVE Data Time-Out
```

Once the transmitted data has been received, the test verifies the status of the receive data packet. This failure is the result of the receive data packet having been received in error:

```
RECEIVE Status Error:
COMPLETE-Bit =1, OK-Bit=0, STATUS-Bits =0000
```

STATUS-Bits breakdown:

Bit #12 Length of error if configured to check length.

Bit #11 CRC error in an aligned frame.

Bit #10 Alignment error (CRC error in a misaligned frame).

Bit #9 Ran out of buffer space - no resources.

Bit #8 DMA Overrun. Failure to acquire the system bus.

Bit #7 Frame too short.

Bit #6 No EOP flag (for Bit stuffing only).

Bit #1 1A Match Bit. When it is zero, the destination address of a received frame matches the IA address. When it is one, the destination address of the received frame does not match the individual address. For example, a multicast or broadcast address sets this bit to a one.

Bit #0 Receive collision. A collision is detected during reception.

Once the data packet has been received and the receive data status verifies, the test verifies that the number of bytes received equals the number of bytes transmitted. This failure is the result of the receive data count and the transmit data count not being equal:

```
RECEIVE Data Transfer Count Error:
Expected =05EA, Actual =003C
```

Upon completion of all the status checks, the test now compares the received data to the transmitted data. This failure results if the data does not verify (compare):

```
Receive Data Miscompare Error:
Address =0000E2C0, Expected =3E3F, Actual =3E3E
```

External Loopback Transceiver - ELBT

Verifies that the 82596 can be operated in the "External Loopback with the LPBK pin activated" mode.

The 82596 has three modes of loopback: Internal Loopback, External Loopback with the LPBK pin activated, and External Loopback with the LPBK pin not activated. The LPBK pin is connected to the accompanying Ethernet Serial Interface (ESI - 82C501AD) chip. The ESI is then connected to the pulse transformer (PE64102), which in turn is connected to the Ethernet Connector.

In External Loopback mode the 82596 transmits and receives simultaneously at a full rate. This allows checking external hardware as well as the serial link to the transceiver interface. The LPBK pin is used to inform the external hardware (ESI) of the establishment of a transmit to receive connection.

The test sets up a data packet (incrementing data pattern) to be transmitted, transmits it, and waits for the reception of the data. Once the data is received, the data is verified to the data transmitted. This transmit to receive loop is performed 32 times. Use the **CF LANC** command to change the loop count.

Command Input:

```
162-Diag>LANC ELBT
```

Response/Messages:

After the command has been issued, the following line is printed:

```
LANC    ELBT: External Loopback Transceiver......... Running --->
```

If all parts of the test are completed correctly, then the test passes:

```
LANC    ELBT: External Loopback Transceiver......... Running ---> PASSED
```

If any failures occur, the following is displayed (more descriptive text then follows):

```
LANC     ELBT: External Loopback Transceiver......... Running ---> FAILED
(error message)
```

Once the data packet has been set up to be transmitted, the test instructs the 82596 (through the Command Unit) to transmit the data packet. This failure is the result of the 82596 completing with a transmit data error. The status bits of the error message display indicate the source of the problem:

```
TRANSMIT Command Completion Status Error:
OK-Bit =0, ABORT-Bit =0, STATUS-Bits =0010
```

STATUS-Bits breakdown:

Bit #6 Late collision. A late collision (a collision after the slot time elapsed) is detected.

Bit #5 No Carrier Sense signal during transmission. Carrier Sense signal is monitored from the end of Preamble transmission until the end of the Frame Check Sequence for TONOCRS = 1 (Transmit On No Carrier Sense Mode); it indicates that transmission has been executed despite a lack of CRS. For TONOCRS = 0 (Ethernet mode), this bit also indicates unsuccessful transmission (transmission stopped when lack of Carrier Sense has been detected).

Bit #4 Transmission unsuccessful (stopped) due to Loss of Clear to Send signal.

Bit #3 Transmission unsuccessful (stopped) due to DMA Underrun; i.e., the system did not supply data for transmission.

Bit #2 Transmission Deferred, i.e., transmission was not immediate due to previous link activity.

Bit #1 Heartbeat Indicator. Indicates that after a previously performed transmission, and before the most recently performed transmission, (Interframe Spacing) the CDT signal was monitored as active. This indicates that the Ethernet Transceiver Collision Detect logic is performing well. The Heartbeat is monitored during Interframe Spacing period.

Bit #0 Transmission attempt was stopped because the number of collisions exceeded the maximum allowable number of retries.

Once the data packet is transmitted successfully, the test waits for four seconds for the receipt of the data. This failure is the result of the timeout (four seconds) expiring:

```
RECEIVE Data Time-Out
```

Once the transmitted data has been received, the test verifies the status of the receive data packet. This failure is the result of the receive data packet having been received in error:

```
RECEIVE Status Error:
COMPLETE-Bit =1, OK-Bit=0, STATUS-Bits =0000
```

STATUS-Bits breakdown:

Bit #12 Length of error if configured to check length.

Bit #11 CRC error in an aligned frame.

Bit #10 Alignment error (CRC error in a misaligned frame).

Bit #9 Ran out of buffer space - no resources.

Bit #8 DMA Overrun. Failure to acquire the system bus.

Bit #7 Frame too short.

Bit #6 No EOP flag (for Bit stuffing only).

Bit #1 IA Match Bit. When it is zero, the destination address of a received frame matches the IA address. When it is one, the destination address of the received frame does not match the individual address. For example, a multicast or broadcast address sets this bit to a one.

Bit #0 Receive collision. A collision is detected during reception.

Once the data packet has been received and the receive data status verifies, the test verifies that the number of bytes received equals the number of bytes transmitted. This failure is the result of the receive data count and the transmit data count not being equal:

```
RECEIVE Data Transfer Count Error:
Expected =05EA, Actual =003C
```

Upon completion of all the status checks, the test now compares the received data to the transmitted data. This failure results if the data does not verify (compare):

```
Receive Data Miscompare Error:
Address =0000E2C0, Expected =3E3F, Actual =3E3E
```

Internal Loopback - ILB

Verifies that the 82596 can be operated in the "Internal Loopback" mode.

The 82596 has three modes of loopback: Internal Loopback, External Loopback with the LPBK pin activated, and External Loopback with the LPBK pin not activated. The LPBK pin is connected to the accompanying Ethernet Serial Interface (ESI - 82C501AD) chip. The ESI is then connected to the pulse transformer (PE64102), which in turn is connected to the Ethernet Connector.

In Internal Loopback mode the 82596 disconnects itself from the serial link and logically connects TXD to RXD and TXC to RXC. The TXC frequency is internally divided by four during internal loopback operation.

The test sets up a data packet (incrementing data pattern) to be transmitted, transmits it, and waits for the reception of the data. Once the data is received, the data is verified to the data transmitted. This transmit to receive loop is performed 32 times. Use the **CF LANC** command to change the loop count.

Command Input:

```
162-Diag>LANC ILB
```

Response/Messages:

After the command has been issued, the following line is printed:

```
LANC    ILB: Internal Loopback...................... Running --->
```

If all parts of the test are completed correctly, then the test passes:

```
LANC    ILB: Internal Loopback...................... Running ---> PASSED
```

If any failures occur, the following is displayed (more descriptive text then follows):

```
LANC     ILB: Internal Loopback...................... Running ---> FAILED
(error message)
```

Once the data packet has been set up to be transmitted, the test instructs the 82596 (through the Command Unit) to transmit the data packet. This failure is the result of the 82596 completing with a transmit data error. The status bits of the error message display indicate the source of the problem:

```
TRANSMIT Command Completion Status Error:
OK-Bit =0, ABORT-Bit =0, STATUS-Bits =0010
```

STATUS-Bits breakdown:

Bit #6 Late collision. A late collision (a collision after the slot time elapsed) is detected.

Bit #5 No Carrier Sense signal during transmission. Carrier Sense signal is monitored from the end of Preamble transmission until the end of the Frame Check Sequence for TONOCRS = 1 (Transmit On No Carrier Sense Mode); it indicates that transmission has been executed despite a lack of CRS. For TONOCRS = 0 (Ethernet mode), this bit also indicates unsuccessful transmission (transmission stopped when lack of Carrier Sense has been detected).

Bit #4 Transmission unsuccessful (stopped) due to Loss of Clear to Send signal.

Bit #3 Transmission unsuccessful (stopped) due to DMA Underrun; i.e., the system did not supply data for transmission.

Bit #2 Transmission Deferred, i.e., transmission was not immediate due to previous link activity.

Bit #1 Heartbeat Indicator. Indicates that after a previously performed transmission, and before the most recently performed transmission, (Interframe Spacing) the CDT signal was monitored as active. This indicates that the Ethernet Transceiver Collision Detect logic is performing well. The Heartbeat is monitored during Interframe Spacing period.

Bit #0 Transmission attempt was stopped because the number of collisions exceeded the maximum allowable number of retries.

Once the data packet is transmitted successfully, the test waits for four seconds for the receipt of the data. This failure is the result of the timeout (four seconds) expiring:

```
RECEIVE Data Time-Out
```

Once the transmitted data has been received, the test verifies the status of the receive data packet. This failure is the result of the receive data packet having been received in error:

```
RECEIVE Status Error:
COMPLETE-Bit =1, OK-Bit=0, STATUS-Bits =0000
```

STATUS-Bits breakdown:

Bit #12 Length of error if configured to check length.

Bit #11 CRC error in an aligned frame.

Bit #10 Alignment error (CRC error in a misaligned frame).

Bit #9 Ran out of buffer space - no resources.

Bit #8 DMA Overrun. Failure to acquire the system bus.

Bit #7 Frame too short.

Bit #6 No EOP flag (for Bit stuffing only).

Bit #1 IA Match Bit. When it is zero, the destination address of a received frame matches the IA address. When it is one, the destination address of the received frame does not match the individual address. For example, a multicast or broadcast address sets this bit to a one.

Bit #0 Receive collision. A collision is detected during reception.

Once the data packet has been received and the receive data status verifies, the test verifies that the number of bytes received equals the number of bytes transmitted. This failure is the result of the receive data count and the transmit data count not being equal:

```
RECEIVE Data Transfer Count Error:
Expected =05EA, Actual =003C
```

Upon completion of all the status checks, the test now compares the received data to the transmitted data. This failure results if the data does not verify (compare):

```
Receive Data Miscompare Error:
Address =0000E2C0, Expected =3E3F, Actual =3E3E
```

Interrupt Request - IRQ

Verifies that the 82596 can assert an interrupt request to the MPU. The 82596 has only one line to signal its interrupt request. The 82596's interrupt request is controlled by the Memory Controller Chip (MCC). The test issues an initialization sequence of the 82596 to occur, upon completion of the initialization the 82596 asserts its interrupt request line to the MPU via the MCC. The test verifies that the appropriate interrupt status is set in the MCC and also that the interrupt status can be cleared.

Command Input:

```
162-Diag>LANC IRQ
```

Response/Messages:

After the command has been issued, the following line is printed:

```
LANC    IRQ: Interrupt Request...................... Running --->
```

If all parts of the test are completed correctly, then the test passes:

```
LANC    IRQ: Interrupt Request...................... Running ---> PASSED
```

If any failures occur, the following is displayed (more descriptive text then follows):

```
LANC     IRQ: Interrupt Request...................... Running ---> FAILED
(error message)
```

Prior to the 82596 initialization sequence launch, the interrupt control register in the MCC is verified against the pretest expected results. This failure is the result of the register contents not verifying against the expected pretest results:

```
LANC Interrupt Control/Status Register Error:
Expected =50, Actual =70
```

Upon completion of the initialization sequence of the 82596, the test verifies the interrupt control register for interrupt status. This failure is the result of the register contents not verifying against the expected post test results (i.e., interrupt status bit not set):

```
LANC Interrupt Control/Status Register Error:
Expected =70, Actual =50
```

Once the interrupt status is verified, the interrupt status is cleared via the ICLR bit in the interrupt control register in the MCC. This failure is the result of the interrupt status bit (INT) in the interrupt control register not clearing:

```
LANC Interrupt Control/Status Register Error:
Expected =50, Actual =70
```

Monitor (Incoming Frames) Mode - MON

Instructs the 82596 to monitor all incoming (receive data) frames. This test is subclassed as a utility test and does not execute when the **LANC** test group is executed. The utility is provided for diagnostic purposes only. Note that no frames are transferred to memory (i.e., 82596 Monitor Mode #3).

This utility, which has no PASS/FAIL message associated with it, executes continuously. You must press the BREAK key to exit (abort).

Command Input:

```
162-Diag>LANC MON
```

Response/Messages:

```
CRCE=0000000 AE=0000000 SF=0000000 RC=0000000 TGB=0000000 TG=0000000
```

Where:

`CRCE` 32-bit count that specifies the number of aligned frames discarded because of a CRC error.

`AE` 32-bit count that specifies the number of frames that are both misaligned (i.e., CRS deasserts on a non-octet boundary) and contain a CRC error.

`SF` 32-bit count that specifies the number of received frames that are shorter than the minimum length.

`RC` 32-bit count that specifies the number of collisions detected during frame reception.

`TGB` 32-bit count that specifies the number of good and bad frames received.

`TG` 32-bit count that specifies the number of good frames received.

The Short Frame counter has priority over CRC, Alignment, and RX Collision counters. Only one of these counters is incremented per frame. For example, if a received frame is both short and collided, only the Short Frame counter is incremented.

Time Domain Reflectometry - TDR

Verifies that Time Domain Reflectometry (TDR) can be executed, and that an error free completion status is returned.

This test activates the TDR feature of the 82596, which is a mechanism to detect open or shorts on the link and their distance from the diagnosing station. The maximum length of the TDR frame is 2048 bits. If the 82596 senses collision while transmitting the TDR frame, it transmits the jam pattern and stops the transmission. The 82596 then triggers an internal timer (STC); the timer is reset at the beginning of transmission and reset if CRS is returned. The timer measures the time elapsed from the start of transmission until an echo is returned. The echo is indicated by Collision Detect going active or a drop in the Carrier Sense signal.

There are four possible results:

1. The Carrier Sense signal does not go active before the counter expires. For a Transceiver that should return Carrier Sense during transmission, this means that there is a problem on the cable between the 82596 and the Transceiver. For a Transceiver that should not return Carrier Sense during transmission, this is normal.
2. The Carrier Sense signal goes active and then inactive before the counter expires. For a Transceiver that should return Carrier Sense during transmission, this means that there is a short on the link.
3. The Collision Detect signal goes active before the counter expires. This means that the link is not properly terminated (an open).
4. The Carrier Sense signal goes active but does not go inactive and Collision Detect does not go active before the counter expires. This is the normal case and indicates that there is no problem on the link.

The distance to the cable failure can be calculated as follows:

Distance = TIME X (Vs / (2 X Fs))

where:

Vs = wave propagation speed on the link (M/s)
Fs = serial clock frequency (Hz)
Accuracy is plus/minus Vs / (2 X Fs)

Note that this test does not execute when the **LANC** test group is executed (**LANC** with no arguments). This test is supplied only for diagnostic purposes. It requires a properly set up Ethernet network (cable).

Command Input:

```
162-Diag>LANC TDR
```

Response/Messages:

After the command has been issued, the following line is printed:

```
LANC    TDR: Time Domain Reflectometry.............. Running --->
```

If all parts of the test are completed correctly, then the test passes:

```
LANC    TDR: Time Domain Reflectometry.............. Running ---> PASSED
```

If any failures occur, the following is displayed (more descriptive text then follows):

```
LANC     TDR: Time Domain Reflectometry.............. Running ---> FAILED
(error message)
```

This failure is the result of TDR command executing with error status:

TDR Command Completion Status Error:
OK-Bit =0

Once the TDR command has completed successfully, the LINK-OK bit is checked in the TDR command packet. This failure is the result of the LINK-OK bit being false (problem with link). The various diagnostic parameters also are displayed with an error message:

```
TDR Command Results Error:
Transceiver Problem       =TRUE or FALSE
Termination Problem       =TRUE or FALSE
Transmission Line Shorted =TRUE or FALSE
Transmit Clock Cycles     =0 to 7FF
```

Additional Error Messages

The following error messages, and descriptions for each, may apply to any or all of the tests within the **LANC** test group.

If the amount memory found during the diagnostics subsystem initialization does not meet the amount of memory needed by the **LANC** test group:

```
Test Initialization Error:
Not Enough Memory, Need =00010000, Actual =000087F0
```

If the control memory address specified by the **LANC** test group configuration parameters is not 16 byte aligned:

```
Test Initialization Error:
Control Memory Address Not 16 Byte Aligned =0000E008
```

The ISCP (Intermediate System Configuration Pointer) indicates the location of the SCB (System Control Block). The CPU loads the SCB address into the ISCP and asserts CA (Channel Attention). This Channel Attention signal causes the 82596 to begin its initialization procedure to get the SCB address from the ISCP. The SCB is the central point through which the CPU and the 82596 exchange control and status information. This failure is the result of the busy byte in the ISCP not becoming clear after one tenth of a second from the issue of the channel attention:

```
LANC Initialization Error:
SCB Read Failure (Channel Attention Signal)
```

During the initialization process of the 82596, the LANC test group initialization function issues an interrupt acknowledge command to the 82596 to acknowledge the completion of the 82596 initialization. This failure is the result of the 82596 command queue not accepting the command:

```
LANC Initialization Error:
LANC Command Unit Command Acceptance Time-Out
```

During the initialization process of the 82596, the **LANC** test group initialization function issues an interrupt acknowledge command to the 82596 to acknowledge the completion of the 82596 initialization. Once the command is accepted by the 82596, the initialization function waits for the 82596 to post status of the completion of the command. This failure is the result of the command timing out from the issue of the command. The timeout value is set to one second:

```
LANC Initialization Error:
LANC Command Unit Interrupt Acknowledge Command Completion Time-Out
```

At the completion of each test in the **LANC** test group the **LANC** error status register (MCC - $FFF42028) is checked for any possible bus error conditions that may have been encountered by the **LANC** while performing DMA accesses to the local bus. This failure is the result of any bus error condition:

```
LANC Error Status Register (DMA Bits) Not Clear =02
```

Prior to issuing a command to the Command Unit of the 82596, the command execution function verifies that the command unit is idle. This failure is the result of the command unit not being in the idle state:

```
LANC Command Unit Not Idle (Busy)
```

Prior to issuing a command to the Receive Unit of the 82596, the receive command execution function verifies that the receive unit is idle. This failure is the result of the receive unit not being in the idle state:

```
LANC Receive Unit Not Idle (Busy)
```

Prior to issuing a command to the Command Unit of the 82596, the command execution function verifies that the command unit does not have any outstanding (pending) interrupt requests. This failure is the result of the command unit having pending interrupt requests:

```
LANC Command Unit Interrupt(s) Pending
```

When a command is issued to the 82596, the command execution function verifies that the 82596 accepted the command. The command execution function waits for one second for this event to occur. This failure is the result of the one second timeout expiring:

```
LANC Command Unit Command Acceptance Time-Out
```

Once a command has been accepted by the 82596, the command execution function waits for the command to complete. The command execution function waits for eight seconds for this event to occur. This failure is the result of the eight second timeout expiring:

```
LANC Command Unit Command Completion Time-Out
```

Once a command has been completed by the 82596, the command execution function waits for the appropriate interrupt status to be posted by the 82596. The command execution function waits for one second for this event to occur. This failure is the result of the one second timeout expiring:

```
LANC Command Unit Interrupt Status Time-Out
```

Once the appropriate interrupt status is set by the 82596, the command execution function issues an interrupt acknowledge command to the command unit of the 82596. Once this command is issued to the 82596, the command execution function waits for one second for the 82596 to post the completion of the interrupt acknowledge command. This failure is the result of the one second timeout expiring:

```
LANC Command Unit Interrupt Acknowledge Command Completion Time-Out
```

When a receive command is issued to the 82596, the receive command execution function verifies that the 82596 accepted the receive command. The receive command execution function waits for one second for this event to occur. This failure is the result of the one second timeout expiring:

```
LANC Receive Unit Command Acceptance Time-Out
```

Once the appropriate interrupt status is set by the 82596, the receive command execution function issues an interrupt acknowledge command to the receive command unit of the 82596. Once this command is issued to the 82596, the receive command execution function waits for one second for the 82596 to post the completion of the interrupt acknowledge command. This failure is the result of the one second timeout expiring:

```
LANC Receive Unit Interrupt Acknowledge Command Completion Time-Out
```

Upon completion of the Configure with Operating Parameters command, the command completion status is verified that it was successful. This failure is the result of an error condition in the completion of the command:

```
Configure Command Completion Status Error:
OK-Bit =0, ABORT-Bit =0
```

Upon completion of the Individual Address Setup command, the command completion status is verified that it was successful. This failure is the result of an error condition in the completion of the command:

```
Individual Address Setup Command Completion Status Error:
OK-Bit =0, ABORT-Bit =0
```

NCR 53C710 SCSI I/O Processor (NCR) Tests

These sections describe the individual NCR 53C710 (SCSI I/O Processor) tests.

Entering **NCR** without parameters causes all **NCR** tests in the order shown in the table below.

To run an individual test, add that test name to the **NCR** command. The individual tests are described in alphabetical order on the following pages. The error message displays following the explanation of a **NCR** test pertain to the test being discussed.

Table 3-10. NCR Test Group

Mnemonic	Description
ACC1	Device Access
ACC2	Register Access
SFIFO	SCSI FIFO
DFIFO	DMA FIFO
LPBK	Loopback
SCRIPTS	SCRIPTs Processor
IRQ	Interrupts

Use the **CF NCR** command to change these parameters:

❏ The "Test Memory Base Address" for the **IRQ** and **SCRIPTS** tests

❏ The "Memory Move Addresses and Byte Count" for the **SCRIPTS** test

Device Access - ACC1

Tests the ability to access the NCR 53C710 device.

1. All device registers are accessed (read) on 8-bit and 32-bit boundaries. (No attempt is made to verify the contents of the registers.)
2. The device data lines are checked by successive writes and reads to the SCRATCH register, by walking a 1 bit through a field of zeros and walking a 0 bit through a field of ones.

If no errors are detected, then the NCR device is reset; otherwise the device is left in the test state.

Command Input:

`162-Diag>`**NCR ACC1**

Response/Messages:

After the command has been issued, the following line is printed:

```
NCR     ACC1: Device Access......................... Running --->
```

If all parts of the test are completed correctly, then the test passes:

```
NCR     ACC1: Device Access......................... Running ---> PASSED
```

If any part of the test fails, then the display appears as follows:

```
NCR     ACC1: Device Access......................... Running ---> FAILED
```

```
NCR/ACC1 Test Failure Data:
(error message)
```

Here, (error message) is one of the following:

```
SCRATCH Register is not initially cleared

Device Access Error:
Address =________, Expected =________, Actual =________

Device Access Error:

Bus Error Information:
               Address ________
                  Data ________
           Access Size __
           Access Type _
    Address Space Code _
         Vector Number ___

Unsolicited Exception:
        Program Counter ________
          Vector Number ___
        Status Register ____
        Interrupt Level _
```

NOTES:

1. All error message data is displayed as hexadecimal values.
2. The Unsolicited Exception information is only displayed if the exception was not a Bus Error.
3. Access Size is displayed in bytes.
4. Access Type is:

0	write
1	read

5. The Address Space Code is:

1	user data
2	user program
5	supervisor data
6	supervisor program
7	MPU space

Register Access - ACC2

Tests the ability to access the NCR 53C710 registers, by checking the state of the registers from a software reset condition and checking their read/write ability. Status registers are checked for initial clear condition after a software reset. Writable registers are written and read with a walking 1 through a field of zeros. If no errors are detected, then the NCR device is reset; otherwise the device is left in the test state.

Command Input:

```
162-Diag>NCR ACC2
```

Response/Messages:

After the command has been issued, the following line is printed:

```
NCR     ACC2: Register Access....................... Running --->
```

If all parts of the test are completed correctly, then the test passes:

```
NCR     ACC2: Register Access....................... Running ---> PASSED
```

If any part of the test fails, then the display appears as follows:

```
NCR     ACC2: Register Access....................... Running ---> FAILED

NCR/ACC2 Test Failure Data:
(error message)
```

Here, (error message) is one of the following:

```
ISTAT Register is not initially cleared

SSTAT0 Register is not initially cleared

SSTAT1 Register is not initially cleared

SSTAT2 Register is not initially cleared

SIEN Register Error:
Address =________, Expected =__, Actual =__

SDID Register Error:
Address =________, Expected =__, Actual =__

SODL Register Error:
Address =________, Expected =__, Actual =__

SXFER Register Error:
Address =________, Expected =__, Actual =__
```

```
SCID Register Error:
Address =________, Expected =__, Actual =__

DSA Register Error:
Address =________, Expected =________, Actual =________

TEMP Register Error:
Address =________, Expected =________, Actual =________

DMA Next Address Error:
Address =________, Expected =________, Actual =________

Register Access Error:

Bus Error Information:
               Address ________
                  Data ________
           Access Size __
           Access Type _
    Address Space Code _
         Vector Number ___

Unsolicited Exception:
        Program Counter ________
          Vector Number ___
        Status Register ____
        Interrupt Level _
```

NOTES:

1. All error message data is displayed as hexadecimal values.
2. The Unsolicited Exception information is only displayed if the exception was not a Bus Error.
3. Access Size is displayed in bytes.
4. Access Type is:

0	write
1	read

5. The Address Space Code is:

1	user data
2	user program
5	supervisor data
6	supervisor program
7	MPU space

DMA FIFO - DFIFO

Tests the ability to write data into the DMA FIFO and retrieve it in the same order as written. The DMA FIFO is checked for an empty condition following a software reset, then the FBL2 bit is set and verified. The FIFO is then filled with 16 bytes of data in the four byte lanes verifying the byte lane full or empty with each write. Next the FIFO is read verifying the data and the byte lane full or empty with each read. If no errors are detected, then the NCR device is reset; otherwise the device is left in the test state.

Command Input:

```
162-Diag>NCR DFIFO
```

Response/Messages:

After the command has been issued, the following line is printed:

```
NCR     DFIFO: DMA FIFO............................ Running --->
```

If all parts of the test are completed correctly, then the test passes:

```
NCR     DFIFO: DMA FIFO............................ Running ---> PASSED
```

If any part of the test fails, then the display appears as follows:

```
NCR     DFIFO: DMA FIFO............................ Running ---> FAILED
```

```
NCR/DFIFO Test Failure Data:
(error message)
```

Here, (error message) is one of the following:

```
DMA FIFO is not initially empty
```

```
DMA FIFO Byte Control not enabled
Address =________, Expected =__, Actual =__
```

```
DMA FIFO Byte Control Error:
Address =________, Expected =__, Actual =__
```

```
DMA FIFO Empty/Full Error:
Address =________, Expected =__, Actual =__
```

```
DMA FIFO Parity Error:
Address =________, Expected =__, Actual =__
DMA FIFO Byte Lane _
```

```
DMA FIFO Error:
Address =________, Expected =__, Actual =__
DMA FIFO Byte Lane _
```

Interrupts - IRQ

Verifies that level 0 interrupts will not generate an interrupt, but will set the appropriate status. The test then verifies that all interrupts (1-7) can be generated and received and that the appropriate status is set.

Command Input:

162-Diag>**NCR IRQ**

Response/Messages:

After the command has been issued, the following line is printed:

```
NCR    IRQ: NCR 53C710 Interrupts.................... Running --->
```

If all parts of the test are completed correctly, then the test passes:

```
NCR     IRQ: NCR 53C710 Interrupts.................... Running --->
PASSED
```

If any part of the test fails, then the display appears as follows:

```
NCR     IRQ: NCR 53C710 Interrupts.................... Running --->
FAILED
```

```
NCR/IRQ Test Failure Data:
(error message)
```

Here, (error message) is one of the following:

```
Test Initialization Error:
Not Enough Memory, Need =________, Actual =________

Test Initialization Error:
Memory Move Byte Count to Large, Max =00ffffff, Requested =________

Test Initialization Error:
Test Memory Base Address Not 32 Bit Aligned =________

SCSI Status Zero "SGE" bit not set
Address =________, Expected =__, Actual =__

Interrupt Status "SIP" bit not set
Address =________, Expected =__, Actual =__

SCSI Status Zero "SGE" bit will not clear
Address =________, Expected =__, Actual =__

Interrupt Status "SIP" bit will not clear
Address =________, Expected =__, Actual =__

Interrupt Control Reg. not initially clear
Address =________, Expected =__, Actual =__
```

```
SCSI Interrupt Enable "SGE" bit not set
Address =________, Expected =__, Actual =__

Interrupt Control "IEN" bit not set
Address =________, Expected =__, Actual =__

Interrupt Status bit did not set
Status: Expected =__, Actual =__
Vector: Expected =__, Actual =__
State : IRQ Level =_, VBR =__

Interrupt Control "INT" bit will not clear
Address =________, Expected =__, Actual =__

SCSI Interrupt Enable Reg. will not mask interrupts
Address =________, Expected =__, Actual =__

Incorrect Vector type
Status: Expected =__, Actual =__
Vector: Expected =__, Actual =__
State : IRQ Level =_, VBR =__

SCSI Interrupt Status:
Expected =__, Actual =__
DMA Interrupt Status:
Expected =__, Actual =__

Unexpected Vector taken
Status: Expected =__, Actual =__
Vector: Expected =__, Actual =__
State : IRQ Level =_, VBR =__

Incorrect Interrupt Level
Level : Expected =_, Actual =_
State : IRQ Level =_, VBR =__

Interrupt did not occur
Status: Expected =__, Actual =__
Vector: Expected =__, Actual =__
State : IRQ Level =_, VBR =__

Interrupt Status bit did not set
Status: Expected =__, Actual =__
Vector: Expected =__, Actual =__
State : IRQ Level =_, VBR =__

Interrupt Control "INT" bit will not clear
Address =________, Expected =__, Actual =__

Bus Error Information:
               Address ________
                  Data ________
           Access Size __
           Access Type _
    Address Space Code _
         Vector Number ___
```

```
Unsolicited Exception:
        Program Counter ________
          Vector Number ___
        Status Register ____
        Interrupt Level _
```

Loopback - LPBK

Checks the Input and Output Data Latches and performs a selection, with the 53C710 executing initiator instructions and the host CPU implementing the target role by asserting and polling the appropriate SCSI signals. The 53C710 Loopback Mode, in effect, lets the chip talk to itself. When the Loopback Enable (SLBE) bit is set in the CTEST4 register, the 53C710 allows control of all SCSI signals When the Loopback Enable (SLBE) bit is set in the CTEST4 register, the 53C710 allows control of all SCSI signals. If no errors are detected, then the NCR device is reset; otherwise the device is left in the test state.

Command Input:

`162-Diag>`**NCR LPBK**

Response/Messages:

After the command has been issued, the following line is printed:

```
NCR     LPBK: Loopback............................. Running --->
```

If all parts of the test are completed correctly, then the test passes:

```
NCR     LPBK: Loopback............................. Running ---> PASSED
```

If any part of the test fails, then the display appears as follows:

```
NCR     LPBK: Loopback............................. Running ---> FAILED
```

```
NCR/LPBK Test Failure Data:
(error message)
```

Here, (error message) is one of the following:

```
No Automatic Clear of 'ADCK' bit in 'CTEST5' Register
```

```
No Automatic Clear of 'BBCK' bit in 'CTEST5' Register
```

```
NCR SCSI Bus Data Lines Error:
Address =________, Expected =__, Actual =__
```

```
DMA Next Address Error:
Address =________, Expected =________, Actual =________
```

```
DMA Byte Counter Error:
Address =________, Expected =________, Actual =________
```

SCRIPTs Processor test - SCRIPTS

Initializes the test structures and makes use of the diagnostic registers for test, as follows:

Verifies that the following registers are initially clear:

SIEN	SCSI Interrupt Enable
DIEN	DMA Interrupt Enable
SSTAT0	SCSI Status Zero
DSTAT	DMA Status
ISTAT	Interrupt Status
SFBR	SCSI First Byte Received

Sets SCSI outputs in high impedance state, disables interrupts using the "MIEN", and sets NCR device for Single Step Mode.

The address of a simple "INTERRUPT instruction" SCRIPT is loaded into the DMA SCRIPTs Pointer register. The SCRIPTs processor is started by hitting the "STD" bit in the DMA Control Register.

Single Step is checked by verifying that ONLY the first instruction executed and that the correct status bits are set. Single Step Mode is then turned off and the SCRIPTs processor started again. The "INTERRUPT instruction" should then be executed and a check for the correct status bits set is made.

The address of the "JUMP instruction" SCRIPT is loaded into the DMA SCRIPTs Pointer register, and the SCRIPTs processor is automatically started. JUMP "if TRUE" (Compare = True, Compare = False) conditions are checked, then JUMP "if FALSE" (Compare = True, Compare = False) conditions are checked.

The "Memory Move instruction" SCRIPT is built in a script buffer to allow the "Source Address", "Destination Address", and "Byte Count" to be changed by use of the "cnfg" command. If a parameter is changed, the only check for validity is the "Byte Count" during test structures initialization.

The "Memory Move" SCRIPT copies the specified number of bytes from the source address to the destination address.

Command Input:

```
162-Diag>NCR SCRIPTS
```

Response/Messages:

After the command has been issued, the following line is printed:

```
NCR    SCRIPTS: NCR 53C710 SCRIPTs Processor......... Running --->
```

If all parts of the test are completed correctly, then the test passes:

```
NCR    SCRIPTS: NCR 53C710 SCRIPTs Processor......... Running ---> PASSED
```

If any part of the test fails, then the display appears as follows:

```
NCR    SCRIPTS: NCR 53C710 SCRIPTs Processor......... Running ---> FAILED

NCR/SCRIPTS Test Failure Data:
(error message)
```

Here, (error message) is one of the following:

```
Test Initialization Error:
Not Enough Memory, Need =________, Actual =________

Test Initialization Error:
Memory Move Byte Count to Large, Max =00ffffff, Requested =________

Test Initialization Error:
Test Memory Base Address Not 32 Bit Aligned =________

SCSI Interrupt Enable Reg. not initially clear
Address =________, Expected =__, Actual =__

DMA Interrupt Enable Reg. not initially clear
Address =________, Expected =__, Actual =__

SCSI Status Zero Reg. not initially clear
Address =________, Expected =__, Actual =__

DMA Status Reg. not initially clear
Address =________, Expected =__, Actual =__

Interrupt Status Reg. not initially clear
Address =________, Expected =__, Actual =__

SCSI First Byte Received Reg. not initially clear
Address =________, Expected =__, Actual =__

SCSI First Byte Received Reg. not set
Address =________, Expected =__, Actual =__

DMA Status "SSI" bit not set
Address =________, Expected =__, Actual =__

Interrupt Status "DIP" bit not set
Address =________, Expected =__, Actual =__

SCSI Status Zero Reg. set during single step
Address =________, Expected =__, Actual =__
```

```
Test Timeout during: INTERRUPT SCRIPTs Test
Address =________, Expected =__, Actual =__

"SIR" not detected during: INTERRUPT SCRIPTs Test
Address =________, Expected =__, Actual =__

Test Timeout during: JUMP SCRIPTs Test
Address =________, Expected =__, Actual =__

"SIR" not detected during: JUMP SCRIPTs Test
Address =________, Expected =__, Actual =__

Jump if "True", and Compare = True; Jump not taken

Jump if "True", and Compare = False; Jump taken

Jump if "False", and Compare = True; Jump taken

Jump if "True", and Compare = False; Jump not taken

Test Timeout during: Memory Move SCRIPTs Test
Address =________, Expected =__, Actual =__

"SIR" not detected during: Memory Move SCRIPTs Test
Address =________, Expected =__, Actual =__
```

SCSI FIFO test - SFIFO

Tests the ability to write data into the SCSI FIFO and retrieve it in the same order as written. The SCSI FIFO is checked for an empty condition following a software reset, then the SFWR bit is set and verified. The FIFO is then filled with 8 bytes of data verifying the byte count with each write. Next the SFWR bit is cleared and the FIFO read verifying the byte count with each read. If no errors are detected, then the NCR device is reset; otherwise the device is left in the test state.

Command Input:

162-Diag>**NCR SFIFO**

Response/Messages:

After the command has been issued, the following line is printed:

```
NCR     SFIFO: SCSI FIFO............................ Running --->
```

If all parts of the test are completed correctly, then the test passes:

```
NCR     SFIFO: SCSI FIFO............................ Running ---> PASSED
```

If any part of the test fails, then the display appears as follows:

```
NCR     SFIFO: SCSI FIFO............................ Running ---> FAILED
```

```
NCR/SFIFO Test Failure Data:
(error message)
```

Here, (error message) is one of the following:

```
SCSI FIFO is not initially empty
```

```
SCSI FIFO writes not enabled
```

```
SCSI FIFO Count Error:
Address =________, Expected =__, Actual =__
```

```
SCSI FIFO Error:
Address =________, Expected =__, Actual =__
```

IndustryPack Interface Controller (IPIC) Tests

These sections describe the individual IndustryPack Interface Controller (IPIC) tests. Entering **IPIC** without parameters causes all IPIC tests to run in the order shown in the table below, except as noted.

To run an individual test, add that test name to the **IPIC** command.

Table 3-11. IPIC Test Group

Mnemonic	Description
ACCESSA	Device Access
ACCESSB	Register Access
Executed only when specified:	
INRPT	Interrupt Control

If more than one IPIC is present, you can use the **CF** command to select the IPIC to test, as shown in this example. Enter the base address of the IPIC you want to test.

Command Input:

```
162-Diag>CF IPIC
```

```
IPIC Configuration Data:
IPIC Base Address =FFFBC000?
```

Read Internal Registers - ACCESSA

Verifies that all of the IPIC ASICs internal registers can be read. It does so by reading the device on byte, word, and long word boundaries. The data returned by these reads is ignored. The test passes if the entire address space occupied by the chip is successfully read. If the test passes, the word "PASSED" is displayed; otherwise the word "FAILED" is displayed. In "verbose" mode, it also displays an error message describing the nature of a failure.

Regardless of the outcome of the testing, the original configuration is maintained afterward.

Command Input:

```
162-Diag>IPIC ACCESSA
```

Response/Messages:

After the command has been issued, the following line is printed:

```
IPIC    ACCESSA: Device Access..... Running --->
```

If read cycles on all selected device registers are completed successfully, then the test passes.

```
IPIC    ACCESSA: Device Access..... Running ---> PASSED
```

If an error occurs while reading any device register, then the test fails.

```
IPIC    ACCESSA: Device Access..... Running ---> FAILED
(error message)
```

Refer to the section *IPIC Error Messages* for a list of the error messages and their meaning.

Write to Internal Registers - ACCESSB

Verifies that internal registers of the IPIC ASIC can be written to and read from. It does so by executing "walking bit" tests on all IPIC memory base address registers and memory size registers. For this, test the register spaces are accessed on long word (32-bit) boundaries.

First, the contents of the registers are saved. Then, zeros are written to the registers and read back for verification. Next, a single bit is written as a "one" and "walked" (shifted), through the field of zeroes verifying each "step". The sequence is then repeated, walking a zero through a field of ones. The test passes if all data patterns written are successfully read. If the test passes, the word "PASSED" is displayed; otherwise the word "FAILED" is displayed. In "verbose" mode, the test also displays an error message describing the nature of a failure.

Regardless of the outcome of the testing, an attempt is made to restore the original configuration afterward.

Command Input:

`162-Diag>`**IPIC ACCESSB**

Response/Messages:

After the command has been issued, the following line is printed:

```
IPIC    ACCESSB: Register Access..... Running --->
```

If all data read matches the data written to the selected device registers, then the test passes.

```
IPIC    ACCESSB: Register Access..... Running ---> PASSED
```

If an error occurs while reading or writing any device register, or data read is not the data written, then the test fails.

```
IPIC    ACCESSB: Register Access..... Running ---> FAILED
```

(error message)

Refer to the section *IPIC Error Messages* for a list of the error messages and their meaning.

Interrupt Control Registers - INRPT

Verifies that the bits in the IPIC Interrupt Control Registers are functioning normally. It does so by configuring each register for level zero, interrupts enabled, and toggling the polarity bit. This sets the INT (interrupt) status bit without generating an interrupt. Next the same steps are repeated for each interrupt level up to and including level seven, expecting and servicing interrupts for every level above zero. The test passes if this entire sequence is successful for all eight IPIC interrupt control registers. If the test passes, the word "PASSED" is displayed, otherwise the word "FAILED" is displayed. In "verbose" mode, the test also displays an error message describing the nature of a failure.

After testing, IPIC registers are returned to their original configuration.

Note

Some IndustryPacks may respond to interrupts generated by the IPIC INRPT test. Therefore, do not run the INRPT test with IndustryPacks installed.

Command Input:

`162-Diag>`**IPIC INRPT**

Response/Messages:

After the command has been issued, the following line is printed:

```
IPIC   INRPT: Interrupt Control..... Running --->
```

If all IPIC Interrupt Control Registers complete the test sequence successfully, then the test passes.

```
IPIC   INRPT: Interrupt Control..... Running ---> PASSED
```

If an error occurs during the test sequence or unexpected results are produced, then the test fails.

```
IPIC   INRPT: Interrupt Control..... Running ---> FAILED
(error message)
```

Refer to the section *IPIC Error Messages* for a list of the error messages and their meaning.

IPIC Error Messages

The IPIC test group error messages generally take the following form:

```
IPIC       ACCESSA: Device Access..... Running ---> FAILED
IPIC Test Failure Data:
Register did not clear, address FFFBC004, expected 00000000, read FFFFFFFF
```

A header message describes which test was executing and announces the "Test Failure Data". Following this, the message displays information that identifies the failure symptom. The next table lists the symptoms.

Table 3-12. IPIC Error Messages

Error Message	Symptom or Cause
Register did not clear, address *a*, expected *e*, read *r*	Read data is not zero, when zero was written.
Register access error, address *a*, expected *e*, read *r*	Read data differs from that written.
Interrupt Control Register did not clear	Test was unable to write zero to the IL2-IL0 (interrupt level) bits or the IEN (interrupt enable) bit in an IPIC Interrupt Control Register. Or INT (interrupt) did not clear when the ICLR (interrupt clear) bit was set.
E/L bit did not set	Test was unable to set (write one) the E/L* (edge/level sensitive) bit in an IPIC Interrupt Control Register.
Interrupt Enable bit did not set	Indicates that the test was unable to set the IEN (interrupt enable) bit in an IPIC Interrupt Control Register.
Interrupt Status bit did not set	Indicates that the INT bit in an IPIC Interrupt Control Register did not set as is the expected result of toggling the PLTY (polarity) bit in the same register.
Unexpected Vector taken	Indicates that an exception occurred with unexpected vector.

Table 3-12. IPIC Error Messages (cont'd)

Error Message	Symptom or Cause
`Incorrect Interrupt Level`	Indicates occurrence of an interrupt of unexpected level.
`Interrupt did not occur`	Indicates that the test gave up waiting for an expected interrupt to occur.
`Interrupt Status bit did not clear`	INT (interrupt) bit in an Interrupt Control Register did not clear when the ICLR (interrupt clear) bit was set
`Status: Expected =`*e*`, Actual =`*r*`,`	Indicates the expected and actual contents of the Interrupt Control Register under test after certain failures occur.
`Vector: Expected =`*e*`, Actual =`*r*`,`	Indicates the expected and actual interrupt vector after certain failures of the **IPIC INRPT** test.
`State: IRQ Level =`*r*`,`	Indicates the level of an interrupt request taken after certain failures of the **IPIC INRPT** test.
`Level: Expected =`*e*`, Actual =`*r*`,`	Indicates the expected and actual interrupt level after certain failures of the **IPIC INRPT** test.
`Testing Register Address = `*a*	Indicates the address of the IPIC register being tested when a failure occurred.

Serial Communication Controller (SCC) (Z85230) Tests

These sections describe the individual Serial Communication Controller (SCC) tests. Entering **SCC** without parameters causes all SCC tests to run in the order shown in the table below, except as noted.

To run an individual test, add that test name to the **SCC** command.

Table 3-13. SCC Test Group

Mnemonic	Description
ACCESS	Device/Register Access
IRQ	Interrupt Request
Executed only when specified:	
BAUDS	Baud Rates
ELPBCK	External Loopback
ILPBCK	Internal Loopback
MDMC	Modem Control

You can use the **CF** command to select the ports to tested. This example uses the **CF** to select ports 1 and 3, skipping 0 and 2.

Command Input:

```
162-Diag>CF SCC
SCC Memory Space Base Address          =FFF45000?
Internal-Loopback/Baud-Rates Port Mask =0000000E? 0A (Bit 0 selects port 0,
                                                     Bit 1 selects port 1, etc.
                                                     See note below.)

External-Loopback/Modem-Control Port Mask=0000000E?
```

Note

These tests number the ports of the Z85C230s starting with the first Z85C230 channel 0 as being port 0, the second Z85C230 channel 0 as being port 2. For MVME162-0*xx* product there are only ports 0 and 1. The MVME162-2*xx* product has four ports: 0, 1, 2 and 3.

The first parameter is the base address space for the Z85C230 devices. This is preset for the MVME162 family and should not be changed.

The next two parameters are the port selection masks. These masks are used during testing to identify which ports are to be tested. The default is to test every port except the console port. The **Internal-Loopback/Baud-Rates Port Mask** is used for the **BAUDS** and **ILPBCK** test suites. The **External-Loopback/Modem-Control Port Mask** is only used for the **ELPBCK** and **MDMC** test suites.

SCC Device/Register Access - ACCESS

This test performs a write/read test on two registers in the Z85C230. This test verifies that the device can be both accessed and that the data paths to the device are functioning.

Command Input:

`162-Diag>`**SCC ACCESS**

Response/Messages:

After the command has been issued, the following line is printed:

```
SCC     ACCESS: Device/Register Access.............. Running --->
```

If all parts of the test are completed correctly, then the test passes:

```
SCC     ACCESS: Device/Register Access.............. Running ---> PASSED
```

If any part of the test fails, then the display appears as follows:

```
SCC     ACCESS: Device/Register Access.............. Running ---> FAILED
```

```
SCC/ACCESS Test Failure Data:
(error message)
```

Refer to the section *SCC Error Messages* for a list of the error messages and their meaning.

SCC Interrupt Request - IRQ

This test verifies that the Z85C230 can generate interrupts to the local processor. This is done using the baud rate zero counter interrupt from the Z85C230.

Command Input:

`162-Diag>`**SCC IRQ**

Response/Messages:

After the command has been issued, the following line is printed:

```
SCC     IRQ: Interrupt Request..................... Running --->
```

If all parts of the test are completed correctly, then the test passes:

```
SCC     IRQ: Interrupt Request..................... Running ---> PASSED
```

If any part of the test fails, then the display appears as follows:

```
SCC     IRQ: Interrupt Request..................... Running ---> FAILED
```

```
SCC/IRQ Test Failure Data:
(error message)
```

Refer to the section *SCC Error Messages* for a list of the error messages and their meaning.

SCC Baud Rates - BAUDS

This test transmits 256 characters at various baud rates. The data is received and compared. If any protocol errors are created or the data is not correct when received, the test failed. The bauds tested are: 1200, 2400, 4800, 9600, 19200, 38400.

Note

Because of the design of the Z85C230, when internal loopback testing is performed, data is still transmitted out of the device on the TxD line. This may cause problems with terminals, modem, printers, and any other device attached.

Command Input:

```
162-Diag>SCC BAUDS
```

Response/Messages:

After the command has been issued, the following line is printed:

```
SCC     BAUDS: Baud Rates......................... Running --->
```

If all parts of the test are completed correctly, then the test passes:

```
SCC     BAUDS: Baud Rates......................... Running ---> PASSED
```

If any part of the test fails, then the display appears as follows:

```
SCC     BAUDS: Baud Rates......................... Running ---> FAILED
```

```
SCC/BAUDS Test Failure Data:
(error message)
```

Refer to the section *SCC Error Messages* for a list of the error messages and their meaning.

SCC External Loopback - ELPBCK

This test transmits 256 characters at 38400 baud. The data is received and compared. If any protocol errors are created or the data is not correct when received, the test failed. This test *does* require an external loopback connector to be installed. For this test TxD and RxD need to be connected in the loopback connector.

Command Input:

`162-Diag>`**SCC ELPBCK**

Response/Messages:

After the command has been issued, the following line is printed:

```
SCC     ELPBCK: External Loopback.................. Running --->
```

If all parts of the test are completed correctly, then the test passes:

```
SCC     ELPBCK: External Loopback.................. Running ---> PASSED
```

If any part of the test fails, then the display appears as follows:

```
SCC     ELPBCK: External Loopback.................. Running ---> FAILED
```

```
SCC/ELPBCK Test Failure Data:
(error message)
```

Refer to the section *SCC Error Messages* for a list of the error messages and their meaning.

SCC Internal Loopback - ILPBCK

This test transmits 256 characters at 38400 baud. The data is received and compared. If any protocol errors are created or the data is not correct when received, the test failed.

Note

Because of the design of the Z85C230, when internal loopback testing is performed, data is still transmitted out of the device on the TxD line. This may cause problems with terminals, modem, printers, and any other device attached.

Command Input:

`162-Diag>`**SCC ILPBCK**

Response/Messages:

After the command has been issued, the following line is printed:

```
SCC     ILPBCK: Internal Loopback.................. Running --->
```

If all parts of the test are completed correctly, then the test passes:

```
SCC     ILPBCK: Internal Loopback.................. Running ---> PASSED
```

If any part of the test fails, then the display appears as follows:

```
SCC     ILPBCK: Internal Loopback.................. Running ---> FAILED

SCC/ILPBCK Test Failure Data:
(error message)
```

Refer to the section *SCC Error Messages* for a list of the error messages and their meaning.

SCC Modem Control - MDMC

This test verifies that the Z85C230 can negate/assert selected modem control lines and that the appropriate input control functions. This test *does* require an external loopback connector to be installed. For this test the following connections need to be made in the loopback connector: DTR connected to DCD, and RTS connected to CTS.

Command Input:

`162-Diag>`**SCC MDMC**

Response/Messages:

After the command has been issued, the following line is printed:

```
SCC     MDMC: Modem Control....................... Running --->
```

If all parts of the test are completed correctly, then the test passes:

```
SCC     MDMC: Modem Control....................... Running ---> PASSED
```

If any part of the test fails, then the display appears as follows:

```
SCC     MDMC: Modem Control....................... Running ---> FAILED
```

```
SCC/MDMC Test Failure Data:
(error message)
```

Refer to the section *SCC Error Messages* for a list of the error messages and their meaning.

SCC Error Messages

The SCC test group error messages generally take the following form:

```
SCC BAUDS: Baud Rates....................... Running ---> FAILED
Transmit/Receive Character Miscompare Error:
Expected =55, Actual =5F
SCC Base Address =FFF45000, Channel =01
Baud Rate =1200
```

The first line of the failure identifies what type of failure occurred. The following line provides additional information about the failure.

Table 3-14. SCC Error Messages

Error Message	Symptom or Cause
Exception, Vector *XX*	Indicates occurrence of an unexpected exception

Data Miscompare Error:
Address =*XXXXXXXX*, Register Index =*XX*
Expected =*XX*, Actual =*XX*
	Indicates that data write does not match data read.

Exception Vector Serviced Error: Expected =*XXX*, Actual =*XXX* Interrupt Level =*X* SCC Base Address =*XXXXXXXX*, Channel =*XX*	Incorrect vector taken or provided during interrupt service
Exception failed to occur, Vector Expected =*XXX* Interrupt Level =*X* SCC Base Address =*XXXXXXXX*, Channel =*XX*	During Interrupt testing, no interrupt was generated or received.
Interrupt Not (Stuck-At) Error: Vector =*XXX*, Interrupt Level =*X* SCC Base Address =*XXXXXXXX*, Channel =*XX*	A preexisting interrupt could not be cleared.
SCC Receiver Error: Status =*XXX* SCC Base Address =*XXXXXXXX*, Channel =*XX* Baud Rate =*XXXX* <Additional error info>	This error indicates a data transmission error. Possible error are; framing, parity, or data overrun
SCC Receiver Error: Status =*XX* Break Sequence detected in the RXD stream SCC Base Address =*XXXXXXXX*, Channel =*XX* Baud Rate =*XXXX*	This error indicates an unexpected break was received during testing.

Table 3-14. SCC Error Messages (Continued)

Error Message	Symptom or Cause
Transmit/Receive Character Miscompare Error: Expected =*XX*, Actual =*XX* SCC Base Address =*XXXXXXXX*, Channel =*XX* Baud Rate =*XXXX*	Indicates that data transmitted does not match data received.
Transmitter Ready Time-Out SCC Base Address =*XXXXXXXX*, Channel =*XX* Baud Rate =*XXXX*	The selected ports transmitter never indicated ready to transmit.
Receiver Ready (Character Available) Time-Out SCC Base Address =*XXXXXXXX*, Channel =*XX* Baud Rate =*XXXX*	The receiver has not received a character in the allotted time.
DTR assertion failed to assert DCD SCC Base Address =*XXXXXXXX*, Channel =*XX*	This error indicates that when DTR was driven, DCD did not follow.
DTR negation failed to negate DCD SCC Base Address =*XXXXXXXX*, Channel =*XX*	
RTS assertion failed to assert CTS SCC Base Address =XXXXXXXX, Channel =*XX*	This error indicates that when RTS was driven, CTS did not follow.
RTS negation failed to negate CTS SCC Base Address =*XXXXXXXX*, Channel =*XX*	

MVME162BUG ENVIRONMENT 4

Introduction

You can use 162Bug to configure certain parameters contained in the Non-Volatile RAM (NVRAM), also known as Battery Backed up RAM (BBRAM). Use the **CNFG** command to change operating parameters of the hardware that are contained in the NVRAM board information block. Use the **ENV** command to change configurable parameters in NVRAM.

The **CNFG** and **ENV** commands are both described in the *Debugging Package for Motorola 68K CISC CPUs User's Manual*. Refer to that manual for general information about their use and capabilities.

This chapter presents information about **CNFG** and **ENV** specific to 162Bug, and describes the parameters that can be configured with the **ENV** command.

Configuring the Board Information Block (CNFG)

The CNFG command is used to display and configure the board information block, which is resident within the NVRAM. Although the factory fills all fields except the IndustryPack fields, only these fields MUST contain correct information:

- MPU clock speed
- Ethernet address
- Local SCSI identifier

The board structure for the MVME162 is as follows:

```
162-Bug>cnfg
Board (PWA) Serial Number = "            "
Board Identifier          = "                "
Artwork (PWA) Identifier  = "                "
MPU Clock Speed           = "    "
Ethernet Address          = 08003E200000
Local SCSI Identifier     = "  "
Parity Memory Mezzanine Artwork (PWA) Identifier  = "        "
Parity Memory Mezzanine (PWA) Serial Number       = "        "
Static Memory Mezzanine Artwork (PWA) Identifier  = "        "
Static Memory Mezzanine (PWA) Serial Number       = "        "
ECC Memory Mezzanine #1 Artwork (PWA) Identifier  = "        "
ECC Memory Mezzanine #1 (PWA) Serial Number       = "        "
ECC Memory Mezzanine #2 Artwork (PWA) Identifier  = "        "
ECC Memory Mezzanine #2 (PWA) Serial Number       = "        "
Serial Port 2 Personality Artwork (PWA) Identifier   = "        "
Serial Port 2 Personality Module (PWA) Serial Number = "        "
IndustryPack A Board Identifier         = "        "
IndustryPack A (PWA) Serial Number      = "        "
IndustryPack A Artwork (PWA) Identifier = "        "
IndustryPack B Board Identifier         = "        "
IndustryPack B (PWA) Serial Number      = "        "
IndustryPack B Artwork (PWA) Identifier = "        "
IndustryPack C Board Identifier         = "        "
IndustryPack C (PWA) Serial Number      = "        "
IndustryPack C Artwork (PWA) Identifier = "        "
IndustryPack D Board Identifier         = "        "
IndustryPack D (PWA) Serial Number      = "        "
IndustryPack D Artwork (PWA) Identifier = "        "
162-Bug>
```

The parameters that are quoted are left-justified character (ASCII) strings padded with space characters, and the quotes (") are displayed to indicate the size of the string. Parameters that are not quoted are considered data strings, and data strings are right-justified. The data strings are padded with zeroes if the length is not met.

Refer to the *MVME162* or *MVME162LX Embedded Controller User's Manual* for the actual location and other information about the board information block. Refer to the *Debugging Package for Motorola 68K CISC CPUs User's Manual* for a description of **CNFG** and examples of its use.

Setting Operational Parameters (ENV)

The **ENV** command allows you to view and configure all MVME162Bug operational parameters stored in Non-Volatile RAM (NVRAM).

Refer to the *Debugging Package for Motorola 68K CISC CPUs User's Manual* for a description of the **ENV** command. Additional information on registers in the VMEchip2 and MCchip that affect these parameters is contained in the *MVME162 Embedded Controller Programmer's Reference Guide*. Listed and described below are the parameters that you can configure using **ENV**.

Configuring MVME162Bug Parameters

The parameters that can be configured using **ENV** are:

```
Bug or System environment [B/S] = B?
```

B — Bug is the mode where no system type of support is displayed. However, system-related items are still available. (Default)

S — System is the standard mode of operation, and is the one defaulted to if NVRAM should fail. This mode is defined in Appendix A in the *Debugging Package for Motorola 68K CISC CPUs User's Manual*.

```
Field Service Menu Enable [Y/N] = N?
```

Y Display the field service menu.

N Do not display the field service menu. (Default)

```
Remote Start Method Switch [G/M/B/N] = B?
```

The Remote Start Method Switch is used when the MVME162 is cross-loaded from another VME-based CPU, to start execution of the cross-loaded program.

G Use the Global Control and Status Register (GCSR) (in theVMEchip2 on MVME162 series modules) to pass and start execution of cross-loaded program.

M Use the Multiprocessor Control Register (MPCR) in shared RAM to pass and start execution of cross-loaded program.

B Use both the GCSR and the MPCR methods to pass and start execution of cross-loaded program. (Default)

N Do not use any Remote Start Method.

```
Probe System for Supported I/O Controllers [Y/N] = Y?
```

Y Accesses will be made to the appropriate system buses (e.g., VMEbus, local MPU bus) to determine the presence of supported controllers. (Default)

N Accesses will not be made to the VMEbus to determine the presence of supported controllers.

```
Negate VMEbus SYSFAIL* Always [Y/N] = N?
```

Y Negate VMEbus SYSFAIL during board initialization.

N Negate VMEbus SYSFAIL after successful completion or entrance into the bug command monitor. (Default)

```
Local SCSI Bus Reset on Debugger Startup [Y/N] = N?
```

Y Local SCSI bus is reset on debugger startup.

N Local SCSI bus is not reset on debugger startup. (Default)

```
Local SCSI Bus Negotiations Type [A/S/N] = A?
```

A Asynchronous negotiations. (Default)

S Synchronous negotiations.

N No negotiations.

```
Industry Pack Reset on Debugger Startup [Y/N] = N?
```

Y Industry Pack(s) is/are reset on debugger startup.

N Industry Pack(s) is/are not reset on debugger startup. (Default)

```
Ignore CFGA Block on a Hard Disk Boot [Y/N] = Y
```

Y Enable the ignorance of the Configuration Area (CFGA) Block (hard disk only). (Default)

N Disable the ignorance of the Configuration Area (CFBA) Block (hard disk only).

```
Auto Boot Enable [Y/N]   = N?
```

Y The Auto Boot function is enabled.

N The Auto Boot function is disabled. (Default)

```
Auto Boot at power-up only [Y/N] = Y?
```

Y Auto Boot is attempted at power up reset only. (Default)

N Auto Boot is attempted at any reset.

```
Auto Boot Controller LUN = 00?
```

Refer to Appendix E in the *Debugging Package for Motorola CISC CPUs User's Manual* for a listing of disk/tape controller modules currently supported by the Bug. The default for this parameter is $0.

```
Auto Boot Device LUN     = 00?
```

Refer to Appendix E in the *Debugging Package for Motorola CISC CPUs User's Manual* for a listing of disk/tape devices currently supported by the Bug. The default for this parameter is $0.

```
Auto Boot Abort Delay    = 15?
```

This is the time in seconds that the Auto Boot sequence will delay before starting the boot. The purpose for the delay is to allow you the option of stopping the boot by use of the Break key. The time value is from 0-255 seconds. The default for this parameter is 15.

```
Auto Boot Default String [NULL for a empty string] = ?
```

You may specify a string (filename) which is passed on to the code being booted. The maximum length of this string is 16 characters. The default for this parameter is the null string.

```
ROM Boot Enable [Y/N] = N?
```

Y The ROMboot function is enabled.

N The ROMboot function is disabled. (Default)

```
ROM Boot at power-up only [Y/N] = Y?
```

Y ROMboot is attempted at power up only. (Default)

N ROMboot is attempted at any reset.

```
ROM Boot Enable search of VMEbus [Y/N] = N?
```

Y VMEbus address space will be searched for a ROMboot module in addition to the usual areas of memory.

N VMEbus address space will not be accessed by ROMboot. (Default)

```
ROM Boot Abort Delay    = 0?
```

This is the time in seconds that the ROMboot sequence will delay before starting the boot. The purpose for the delay is to allow you the option of stopping the boot by use of the Break key. The time value is from 0-255 seconds. The default for this parameter is 0 seconds.

```
ROM Boot Direct Starting Address = FF800000?
```

This is the first location tested when the Bug searches for a ROMboot Module. The default address is $FF800000.

```
ROM Boot Direct Ending Address   = FFDFFFFC?
```

This is the last location tested when the Bug searches for a ROMboot Module. The default address is $FFDFFFFC.

```
Network Auto Boot Enable [Y/N]   = N?
```

Y The Network Auto Boot function is enabled.

N The Network Auto Boot function is disabled. (Default)

```
Network Auto Boot at power-up only [Y/N] = Y?
```

Y Network Auto Boot is attempted at power up reset only. (Default)

N Network Auto Boot is attempted at any reset.

```
Network Auto Boot Controller LUN = 00?
```

Refer to Appendix G in the *Debugging Package for Motorola CISC CPUs User's Manual* for a listing of disk/tape controller modules currently supported by the Bug. The default for this parameter is $0.

```
Network Auto Boot Device LUN     = 00?
```

Refer to Appendix G in the *Debugging Package for Motorola CISC CPUs User's Manual* for a listing of disk/tape controller modules currently supported by the Bug. The default for this parameter is $0.

```
Network Auto Boot Abort Delay    = 5?
```

This is the time in seconds that the Network boot sequence will delay before starting the boot. The purpose for the delay is to allow you the option of stopping the boot by use of the Break key. The time value is from 0-255 seconds. The default for this parameter is 5 seconds.

```
Network Auto Boot Configuration Parameters Pointer (NVRAM) = 00000000?
```

This is the address where the network interface configuration parameters are to be saved/retained in NVRAM; these parameters are the necessary parameters to perform an unattended network boot.

```
Memory Search Starting Address   = 00000000?
```

This is where the Bug begins to search for a work page (a 64KB block of memory) to use for vector table, stack, and variables. This must be a multiple of the debugger work page, modulo $10000 (64KB). In a multi-16X environment, each MVME16X board could be set to start its work page at a unique address so as to allow multiple debuggers to operate simultaneously. The default Memory Search Starting Address is $00000000.

```
Memory Search Ending Address     = 00100000?
```

This is the top limit of the Bug's search for a work page. If a contiguous block of memory, 64KB in size, is not found in the range specified by the Memory Search Starting Address and the Memory Search Ending Address parameters, then the bug will place its work page in the onboard static RAM on the MVME16X. The default Memory Search Ending Address is the calculated size of local memory.

```
Memory Search Increment Size     = 00010000?
```

This multi-CPU feature is used to offset the location of the Bug work page. This must be a multiple of the debugger work page, modulo $10000 (64KB). Typically, the Memory Search Increment Size is the product of the CPU number and size of the Bug work page. For example, the Memory Search Increment Size for the first CPU would be $0 (0 x $10000), the second CPU would be $10000 (1 x $10000), etc. The default is $10000.

```
Memory Search Delay Enable [Y/N] = N?
```

Y There will be a delay before the Bug begins its search for a work page. This delay could be used to allow time for some other MVME16X in the system to configure its address decoders.

N There will be no delay before the Bug begins its search for a work page. (Default)

```
Memory Search Delay Address      = FFFFD20F?
```

This is the MVME162 GCSR GPCSR0 address as accessed through VMEbus A16 space and assumes the MVME162 GRPAD (group address) and BDAD (board address within group) switches are set to "on". The byte-wide value is initialized to $FF by MVME162 hardware after a System or Power-on Reset. In a multi-162 environment, where the work pages of several Bugs will reside in the memory of the primary MVME162, the non-primary CPUs will wait for the data at the Memory Search Delay Address to be set to $00, $01, or $02. Refer to the *Memory Requirements* section in Chapter 1 for value definitions before locating their work page in the memory of the primary CPU.

```
Memory Size Enable [Y/N]         = Y?
```

Y Memory will be sized for Self Test diagnostics. (Default)

N Memory will not be sized for Self Test diagnostics.

```
Memory Size Starting Address     = 00000000?
```

The default Starting Address is $0.

```
Memory Size Ending Address       = 00100000?
```

The default Ending Address is the calculated size of local memory.

Note

Memory Configuration Defaults:
The default configuration for Dynamic RAM mezzanine boards will position the mezzanine with the largest memory size to start at the address selected with the "ENV" parameter "Base Address of Dynamic Memory". The Base Address parameter defaults to 0. The smaller sized mezzanine will follow immediately above the larger in the memory map. If mezzanines of the same size and type are present, the first (closest to the board) is mapped to the selected base address. If mezzanines of same size but with different type (parity and ECC) are present, the parity type will be mapped to the selected base address and the ECC type mezzanine will follow. The SRAM does not default to a location in the memory map that is contiguous with Dynamic RAM.

```
Base Address of Dynamic Memory     = 00000000?
```

This is the beginning address of Dynamic Memory (Parity and/or ECC type memory). It must be a multiple of the Dynamic Memory board size, starting with 0. The default for this parameter is $0.

```
Size of Parity Memory     = 00100000?
```

This is the size of the Parity type dynamic RAM mezzanine, if any. The default is the calculated size of the Dynamic memory mezzanine board.

```
Size of ECC Memory Board #0      = 00000000?
```

This is the size of the first ECC type memory mezzanine. The default is the calculated size of the memory mezzanine.

```
Size of ECC Memory Board #1      = 00000000?
```

This is the size of the second ECC type memory mezzanine. The default is the calculated size of the memory mezzanine.

```
Base Address of Static Memory    = FFE00000?
```

This is the beginning address of SRAM. The default for this parameter is FFE00000 for the onboard 128KB SRAM (on MVME162LX) or 512KB (on MVME162), or E1000000 for the 2MB SRAM mezzanine. If only 2MB SRAM is present, it defaults to address 00000000.

```
Size of Static Memory            = 00080000?
```

This is the size of the SRAM type memory present. The default is the calculated size of the onboard SRAM or an SRAM type mezzanine.

Configuring the VMEbus Interface

ENV asks the following series of questions to set up the VMEbus interface for the MVME162 series modules. You should have a working knowledge of the VMEchip2 as given in the *MVME162* or *MVME162LX Embedded Controller Programmer's Reference Guide* in order to perform this configuration. Also included in this series are questions for setting ROM and FLASH access time.

The slave address decoders are used to allow another VMEbus master to access a local resource of the MVME162. There are two slave address decoders set. They are set up as follows:

```
Slave Enable #1 [Y/N] = Y?
```

Y	Yes, set up and enable the Slave Address Decoder #1.(Default)
N	Do not set up and enable the Slave Address Decoder #1.

```
Slave Starting Address #1 = 00000000?
```

This is the base address of the local resource that is accessible by the VMEbus. The default is the base of local memory, $0.

```
Slave Ending Address #1   = 000FFFFF?
```

This is the ending address of the local resource that is accessible by the VMEbus. The default is the end of calculated memory.

```
Slave Address Translation Address #1 = 00000000?
```

This register will allow the VMEbus address and the local address to be different. The value in this register is the base address of local resource that is associated with the starting and ending address selection from the previous questions. The default is 0.

```
Slave Address Translation Select #1  = 00000000?
```

This register defines which bits of the address are significant. A logical one "1" indicates significant address bits, logical zero "0" is non- significant. The default is 0.

```
Slave Control #1 = 03FF?
```

This defines the access restriction for the address space defined with this slave address decoder. The default is $03FF.

```
Slave Enable #2 [Y/N] = N?
```

Y	Yes, set up and enable the Slave Address Decoder #2.

N	Do not set up and enable Slave Address Decoder #2. (Default)

```
Slave Starting Address #2 = 00000000?
```

This is the base address of the local resource that is accessible by the VMEbus. The default is 0.

```
Slave Ending Address #2   = 00000000?
```

This is the ending address of the local resource that is accessible by the VMEbus. The default is 0.

```
Slave Address Translation Address #2 = 00000000?
```

This register will allow the VMEbus address and the local address to be different. The value in this register is the base address of local resource that is associated with the starting and ending address selection from the previous questions. The default is 0.

```
Slave Address Translation Select #2  = 00000000?
```

This register defines which bits of the address are significant. A logical one "1" indicates significant address bits, logical zero "0" is non- significant. The default is 0.

```
Slave Control #2 = 0000?
```

This defines the access restriction for the address space defined with this slave address decoder. The default is $0000.

```
Master Enable #1 [Y/N] = Y?
```

Y	Yes, set up and enable the Master Address Decoder #1. (Default)
N	Do not set up and enable the Master Address Decoder #1.

```
Master Starting Address #1 = 02000000?
```

This is the base address of the VMEbus resource that is accessible from the local bus. The default is the end of calculated local memory, unless memory is less than 16MB, then this register will always be set to 01000000.

```
Master Ending Address #1   = EFFFFFFF?
```

This is the ending address of the VMEbus resource that is accessible from the local bus. The default is the end of calculated memory.

```
Master Control #1 = 0D?
```

This defines the access characteristics for the address space defined with this master address decoder. The default is $0D.

```
Master Enable #2 [Y/N] = N?
```

Y Yes, set up and enable the Master Address Decoder #2.

N Do not set up and enable the Master Address Decoder #2. (Default)

```
Master Starting Address #2 = 00000000?
```

This is the base address of the VMEbus resource that is accessible from the local bus. The default is $00000000.

```
Master Ending Address #2   = 00000000?
```

This is the ending address of the VMEbus resource that is accessible from the local bus. The default is $00000000.

```
Master Control #2 = 00?
```

This defines the access characteristics for the address space defined with this master address decoder. The default is $00.

```
Master Enable #3 [Y/N] = Y?
```

Y Yes, set up and enable the Master Address Decoder #3. This is the default if the board contains less than 16MB of calculated RAM.

N Do not set up and enable the Master Address Decoder #3. This is the default for boards containing at least 16MB of calculated RAM.

```
Master Starting Address #3 = 00000000?
```

This is the base address of the VMEbus resource that is accessible from the local bus. If enabled, the value is calculated as one more than the caluclated size of memory. If not enabled, the default is $00000000.

```
Master Ending Address #3   = 00000000?
```

This is the ending address of the VMEbus resource that is accessible from the local bus. If enabled, the default is $00FFFFFF, otherwise $00000000.

```
Master Control #3 = 00?
```

This defines the access characteristics for the address space defined with this master address decoder. If enabled, the default is $3D, otherwise $00.

```
Master Enable #4 [Y/N] = N?
```

Y Yes, set up and enable the Master Address Decoder #4.

N Do not set up and enable the Master Address Decoder #4. (Default)

```
Master Starting Address #4 = 00000000?
```

This is the base address of the VMEbus resource that is accessible from the local bus. The default is $0.

```
Master Ending Address #4   = 00000000?
```

This is the ending address of the VMEbus resource that is accessible from the local bus. The default is $0.

```
Master Address Translation Address #4 = 00000000?
```

This register will allow the VMEbus address and the local address to be different. The value in this register is the base address of VMEbus resource that is associated with the starting and ending address selection from the previous questions. The default is 0.

```
Master Address Translation Select #4  = 00000000?
```

This register defines which bits of the address are significant. A logical one "1" indicates significant address bits, logical zero "0" is non- significant. The default is 0.

```
Master Control #4 = 00?
```

This defines the access characteristics for the address space defined with this master address decoder. The default is $00.

```
Short I/O (VMEbus A16) Enable [Y/N] = Y?
```

Y Yes, Enable the Short I/O Address Decoder. (Default)

N Do not enable the Master Address Decoder.

```
Short I/O (VMEbus A16) Control      = 01?
```

This defines the access characteristics for the address space defined with the Short I/O address decoder. The default is $01.

```
F-Page (VMEbus A24) Enable [Y/N]    = Y?
```

Y Yes, Enable the F-Page Address Decoder. (Default)

N Do not enable the F-Page Address Decoder.

```
F-Page (VMEbus A24) Control         = 02?
```

This defines the access characteristics for the address space defined with the F-Page address decoder. The default is $02.

```
ROM Access Time Code                = 03?
```

This defines the ROM access time. The default is $03, which sets an access time of 180 nS.

```
FLASH Access Time Code              = 02?
```

This defines the FLASH access time. The default is $02, which sets an access time of 140 nS.

```
MCC Vector Base               = 05?
VMEC2 Vector Base #1          = 06?
VMEC2 Vector Base #2          = 07?
```

These parameters are the base interrupt vector for the component specified. (Default: MCC = $05, VMEchip2 Vector 1 = $06, VMEchip2 Vector 2 = $07.)

```
VMEC2 GCSR Group Base Address = D2?
```

This parameter specifies the group address ($FFFFXX00) in Short I/O for this board. (Default is $D2.)

```
VMEC2 GCSR Board Base Address = 00?
```

This parameter specifies the base address ($FFFFD2XX) in Short I/O for this board. (Default is $00.)

```
VMEbus Global Time Out Code   = 01?
```

This controls the VMEbus timeout when systems controller.
(Default $01 = 64 µs.)

```
Local Bus Time Out Code       = 02?
```

This controls the local bus timeout.
(Default $02 = 256 µs.)

```
VMEbus Access Time Out Code   = 02?
```

This controls the local bus to VMEbus access timeout.
(Default $02 = 32 ms.)

Configuring the IndustryPacks

ENV asks the following series of questions to set up IndustryPacks (IP) on MVME162 modules.

The *MVME162* or *MVME162LX Embedded Controller Programmer's Reference Guide* describes the base addresses and the IP register settings. Refer to that manual for information on setting base addresses and register bits.

```
IP A Base Address                = 00000000?
IP B Base Address                = 00000000?
IP C Base Address                = 00000000?
IP D Base Address                = 00000000?
```

Base address for mapping IP modules. Only the upper 16 bits are significant.

```
IP D/C/B/A Memory Size           = 00000000?
```

Define the memory size requirements for the IP modules:

Bits	IP	Register Address
31-24	D	FFFBC00F
23-16	C	FFFBC00E
15-08	B	FFFBC00D
07-00	A	FFFBC00C

```
IP D/C/B/A General Control       = 00000000?
```

Define the general control requirements for the IP modules:

Bits	IP	Register Address
31-24	D	FFFBC01B
23-16	C	FFFBC01A
15-08	B	FFFBC019
07-00	A	FFFBC018

```
IP D/C/B/A Interrupt 0 Control = 00000000?
```

Define the interrupt control requirements for the IP modules channel 0:

Bits	IP	Register Address
31-24	D	FFFBC016
23-16	C	FFFBC014
15-08	B	FFFBC012
07-00	A	FFFBC010

```
IP D/C/B/A Interrupt 1 Control = 00000000?
```

Define the interrupt control requirements for the IP modules channel 1:

Bits	IP	Register Address
31-24	D	FFFBC017
23-16	C	FFFBC015
15-08	B	FFFBC013
07-00	A	FFFBC011

Caution

Before environment parameters are saved in the NVRAM, a warning message will appear if the user has specified environment parameters which will cause an overlap condition. The important information about each configurable element in the memory map is displayed, showing where any overlap conditions exist. This will allow the user to quickly identify and correct an undesirable configuration before it is saved.

ENV warning example:

```
WARNING: Memory MAP Overlap Condition Exists

S-Address  E-Address  Enable  Overlap  M-Type  Memory-MAP-Name
$00000000  $FFFFFFFF  Yes     Yes      Master  Local Memory (Dynamic RAM)
$FFE00000  $FFE7FFFF  Yes     Yes      Master  Static RAM
$01000000  $EFFFFFFF  Yes     Yes      Master  VMEbus Master #1
$00000000  $00000000  No      No       Master  VMEbus Master #2
$00000000  $00FFFFFF  Yes     Yes      Master  VMEbus Master #3
$00000000  $00000000  No      No       Master  VMEbus Master #4
$F0000000  $FF7FFFFF  Yes     Yes      Master  VMEbus F Pages (A24/A32)
$FFFF0000  $FFFFFFFF  Yes     Yes      Master  VMEbus Short I/O (A16)
$FF800000  $FFBFFFFF  Yes     Yes      Master  Flash/PROM
$FFF00000  $FFFEFFFF  Yes     Yes      Master  Local I/O
$00000000  $00000000  No      No       Master  Industry Pack A
$00000000  $00000000  No      No       Master  Industry Pack B
$00000000  $00000000  No      No       Master  Industry Pack C
$00000000  $00000000  No      No       Master  Industry Pack D
$00000000  $00000000  No      No       Slave   VMEbus Slave #1
$00000000  $00000000  No      No       Slave   VMEbus Slave #2
```

Index

Numerics

162Bug
- general information 1-1
- implementation of 1-6
- installing and starting up 1-6
- stack 1-9

82596 LANC chip 3-100

A

ACC1 test (NCR) 3-127
ACC2 test (NCR) 3-129
ACCESS test (SCC) 3-147
ACCESSA test (IPIC) 3-140
ACCESSA test (MCC) 3-26
ACCESSB test (IPIC) 3-141
ACCESSB test (MCC) 3-27
additional MCC test group messages 3-47
addressing memory 3-3
ADJ test (MCC) 3-28
ADR test (RAM) 3-3
ADR test (RTC) 3-19
AEM command 2-5
Alternating Ones/Zeros (RAM) - ALTS 3-5
ALTS test (RAM) 3-5
Append Error Messages Mode - command AEM 2-5
assertion 1-12
Auto Boot Enable 4-4

B

base address of dynamic memory 4-8
base address of IndustryPacks 4-14
Battery Backed-Up SRAM (RTC) - RAM 3-23
BAUDS test (SCC) 3-149
BBRAM 4-1
- addressing (RTC) - ADR 3-19
- configuring 4-1

binary number 1-12
Bit Toggle (RAM) - BTOG 3-6
board
- address (BDAD) switches 4-7
- information block 4-1
- structure 4-1

BTOG test (RAM) 3-6
Build Default Tables (MMU) - TBLBLD 3-66
byte 1-12

C

cache tests (DCAC) 3-57
CBIT 3-50
CEM command 2-5
CF command 2-5
Check-Bit DRAM - CBIT 3-50
Chip Self Test (LANC) - CST 3-102
Clear Error Counters - command ZE 2-10
Clear Error Messages - command CEM 2-5
Clear On Compare test (VME2) 3-94
clearing diagnostic 2-5
CLK test (RTC) 3-21
clock function (RTC) - CLK 3-21
CNFG command 4-1

Code Execution/Copy (RAM) - CODE 3-8
CODE test (RAM) 3-8
command entry 2-3
configuration parameters 2-5
configuring
- 162Bug environment 4-1
- base address of Industry Packs 4-14
- BBRAM 4-1
- board information block (CNFG) 4-1
- hardware 1-6
- IndustryPacks 4-14
- MVME162bug parameters 4-2
- NVRAM 4-1
- VMEbus interface 4-9

conventions 1-12
Copyback mode (DCAC) 3-58
CST test (LANC) 3-102

D

Data Cache Copyback - DCAC_CB 3-58
Data Cache Registers - DCAC_RD 3-60
data cache tests (DCAC) 3-57
Data Cache Writethrough - DCAC_WT 3-61
Data Patterns (RAM)- PATS 3-9
DCAC tests 3-57
DCAC_CB 3-58
DCAC_CB test 3-58
DCAC_RD 3-60
DCAC_RD test 3-60
DCAC_WT 3-61
DE command 2-5
decimal number 1-12
DEM command 2-6
description of 162Bug 1-1
design features 2-1
Device Access (MCC) - ACCESSA 3-26
Device Access (NCR) - ACC1 3-127
DFIFO test (NCR) 3-131
DIAG test (LANC) 3-104
Diagnose Internal Hardware test (LANC) -DIAG 3-104
diagnostic
- facilities 1-9
- firmware 2-1
- test groups 3-1
- utilities 2-4

directories 2-3, 2-10
- switching 1-1, 1-9

Display Error Counters - command DE 2-5
Display Error Messages - command DEM 2-6
Display Pass Count - command DP 2-6
Display Table Search (MMU) - DISPSRCH 3-65
Display/Revise Self Test Mask - command MASK 2-9
DISPSRCH test (MMU) 3-65
DMA FIFO (NCR) - DFIFO 3-131
documentation, related 1-11
DP command 2-6
DRAM test (MCC) - DRAM 3-32
Dump Configuration/Registers test (LANC) -DUMP 3-106
DUMP test (LANC) 3-106

E

ECC Memory Board (MCECC) Tests 3-48
ELBC test (LANC) 3-107
ELBT test (LANC) 3-111
ELPBCK test (SCC) 3-150
ENV command 4-1, 4-2
environmental parameters 4-1
error counters 2-5, 2-10
error detection (RAM) 3-11
error messages 2-6
- additional MCC 3-47
- appending to 2-5
- clearing diagnostic 2-5
- from IPIC tests 3-144
- from LANC tests 3-122

SCC 3-153
error scrubbing 3-55
error, multi-bit 3-53
error, single-bit 3-54
Ethernet 3-100
Exceptions - EXCPTN 3-52
EXCPTN 3-52
External Loopback Cable test (LANC) - ELBC 3-107
External Loopback Transceiver test (LANC) - ELBT 3-111

F
FAST bit test (MCC) - FAST 3-29
FIFO test (NCR) 3-138
firmware 2-1
flow diagram
of 162Bug Board Operational Mode 1-2
of 162Bug System Operational Mode 1-4

G
GCSR GPCSR0 4-7
general commands 2-4
group address (GRPAD) switches 4-7

H
hardware 1-6
HE command 2-1, 2-2, 2-6
headers 1-6
Help - command HE 2-6
Help Extended - command HEX 2-6
help screen 2-1, 2-2, 2-6
HEX command 2-6
hexadecimal character 1-12

I
I/O processor tests 3-126
ILB test (LANC) 3-114
ILPBCK test (SCC) 3-151
Industry Pack configuration
general control register 4-14
interrupt control registers 4-15
memory size 4-14
IndustryPack Interface Controller (IPIC) tests 3-139, 3-146
INRPT test (IPIC) 3-142
installing and starting up 162Bug 1-6
internal cache tests (DCAC) 3-57
internal error message buffer
appending error messages to 2-5
Internal Loopback test (LANC) - ILB 3-114
Interrupt Request test (LANC) - IRQ 3-117
Interrupt Stack Pointer (ISP) 1-9
interrupts, software
polled 3-81
priority 3-85
processor 3-83
Invalid Page test (MMU) - INVPAGE 3-74
INVPAGE test (MMU) 3-74
IPIC Interrupt Control Registers - INRPT 3-142
IPIC tests 3-139, 3-146
error messages 3-144
interrupt control registers 3-142
read registers 3-140
write to registers 3-141
IRQ test (LANC) 3-117
IRQ test (NCR) 3-132
IRQ test (SCC) 3-148

J
jumpers 1-6

L
LA command 2-8
LAN Coprocessor for Ethernet (LANC) Tests 3-100
LANC error messages 3-122
LANC tests 3-100

CST 3-102
DIAG 3-104
DUMP 3-106
ELBC 3-107
ELBT 3-111
ILB 3-114
IRQ 3-117
MON 3-119
TDR 3-120
LC command 2-8
LE command 2-8
LF command 2-8
Line Feed Suppression Mode - Prefix LF 2-8
LN command 2-9
Local Parity Memory Error Detection (RAM) - PED 3-11
local RAM tests 3-2
longword 1-12
Loop Always Mode - Prefix LA 2-8
Loop Non-Verbose Mode - Prefix LN 2-9
Loopback test (NCR) - LPBK 3-134
Loop-Continue Mode - Prefix LC 2-8
Loop-On-Error Mode - Prefix LE 2-8
LPBK test (NCR) 3-134

M

Mapped ROM Read test (MMU) - MAPROM 3-71
MAPROM test (MMU) 3-71
MASK command 2-9
master address decoders 4-10
MBE 3-53
MC68040 Internal Cache (DCAC) tests 3-57
MCC error messages 3-47
MCC tests 3-24
ACCESSA 3-26
ACCESSB 3-27
ADJ 3-28
DRAM 3-32
FAST 3-29
MPUCS 3-30
PCLK 3-31
TMRnA 3-33
TMRnB 3-35
TMRnC 3-37
TMRnD 3-38
TMRnE 3-40
VBR 3-42
WDTMRA 3-43
WDTMRB 3-44
WDTMRC 3-46
MCECC tests 3-48
MDMC test (SCC) 3-152
memory
base address of dynamic 4-8
memory addressing (RAM) - ADR 3-3
memory board tests 3-48
Memory Controller Chip (MCC) tests 3-24
Memory Management Unit (MMU) tests 3-63
Memory Refresh Testing (RAM) - REF 3-15
memory requirements 1-8
MK48T08 tests 3-23
MK48T0x (RTC) tests 3-18
MMU tests 3-63
DISPSRCH 3-65
INVPAGE 3-74
MAPROM 3-71
MODPAGE 3-73
RP 3-69
TBLBLD 3-66
TBLVERF 3-67
TC 3-68
USEDPAGE 3-72
WALK 3-70
WPPAGE 3-75
MMU translation tables 3-66, 3-67, 3-70
Modified Page test (MMU) - MODPAGE 3-73
MODPAGE test (MMU) 3-73

MON test (LANC) 3-119
Monitor (Incoming Frames) Mode test (LANC) - MON 3-119
monitor start-up 2-2
MPU Clock Speed (MCC) - MPUCS 3-30
MPUCS test (MCC) 3-30
Multi-Bit-Error - MBE 3-53
MVME162Bug parameters 4-2

N

NCR 53C710 SCSI I/O Processor (NCR) Tests 3-126
  ACC1 3-127
  ACC2 3-129
  DFIFO 3-131
  FIFO 3-138
  IRQ 3-132
  LPBK 3-134
  SCRATCH register 3-127
  SCRIPTS 3-135
NCR tests 3-126
negation 1-12
Network Auto Boot Enable 4-5
No Clear On Compare test (VME2) 3-92
Non-Verbose Mode - Prefix NV 2-9
NV command 2-9
NVRAM
  configuring 4-1
  parameters in 4-2

O

Overflow Counter (VME2) - TMRH, TMRI 3-96

P

page test (MMU) 3-72, 3-73, 3-74, 3-75
pass count 2-6, 2-11
PATS test (RAM) 3-9
PCLK test (MCC) 3-31
PED test (RAM) 3-11
PERM test (RAM) 3-13
Permutations (RAM) - PERM 3-13
Prescaler Clock (MCC) - PCLK 3-31
Prescaler Clock Adjust (VME2) - TMRC 3-90
Prescaler Clock Adjust test (MCC) - ADJ 3-28

Q

Quick Write/Read (RAM) - QUIK 3-14
QUIK test (RAM) 3-14

R

RAM test (RTC) 3-23
RAM tests 3-2
RAM/SRAM tests (RAM)
  ADR 3-3
  ALTS 3-5
  BTOG 3-6
  CODE 3-8
  PATS 3-9
  PED 3-11
  PERM 3-13
  QUIK 3-14
  REF 3-15
  RNDM 3-17
Random Data (RAM) - RNDM 3-17
read IPIC registers - ACCESSA 3-140
read-write (RAM) 3-14
REF test (RAM) 3-15
REGA test (VME2) 3-77
REGB test (VME2) 3-79
Register Access (MCC) - ACCESSB 3-27
Register Access (NCR) - ACC2 3-129
Register Access (VME2) - REGA 3-77
register display 3-60
Register Walking Bit (VME2) - REGB 3-79
RNDM test (RAM) 3-17
ROM Boot Enable 4-5
ROM read (MMU) 3-71
ROMboot 1-8
Root Pointer registers (MMU) 3-69
RP register test (MMU) - RP 3-69

RTC tests 3-18
- ADR 3-19
- CLK 3-21
- RAM 3-23

S

SBE 3-54
SCC Baud Rates - BAUDS 3-149
SCC Device/Register Access - ACCESS 3-147
SCC Error Messages 3-153
SCC External Loopback - ELPBCK 3-150
SCC Internal Loopback - ILPBCK 3-151
SCC Interrupt Request - IRQ 3-148
SCC Modem Control - MDMC 3-152
SCC tests 3-146
SCRATCH register test (NCR) 3-126, 3-127
SCRIPTS test (NCR) 3-135
SCRUB 3-55
Scrubbing - SCRUB 3-55
SCSI FIFO (NCR) - SFIFO 3-138
SCSI I/O processor tests 3-126
SCSI specification 1-12
SD command 1-1, 1-9, 2-10
SE command 2-10
Self Test - command ST 2-10
Self Test mask 2-9
Serial Communication Controller (SCC) (Z85230) Tests 3-146
setting operational parameters (ENV) 4-2
Single-Bit-Error - SBE 3-54
slave address decoders 4-9
Software Interrupts (VME2) - SWIA 3-81
Software Interrupts (VME2) - SWIB 3-83
Software Interrupts Priority (VME2) - SWIC 3-85
SRAM tests 3-2
ST command 2-10
start-up 1-6
- monitor 2-2

static RAM tests 3-2
static variable space 1-9
Stop-On-Error Mode - Prefix SE 2-10
SWIA test (VME2) 3-81
SWIB test (VME2) 3-83
SWIC test (VME2) 3-85
Switch Directories - command SD 2-10
switching 1-1, 1-9
system console 1-8
system controller function 1-7
system startup 2-1

T

Tablewalk Mapped Pages (MMU) - WALK 3-70
TACU test (VME2) 3-87
TBLBLD test (MMU) 3-66
TBLVERF test (MMU) 3-67
TC Register test (MMU) - TC 3-68
TDR test (LANC) 3-120
terminology 1-12
test directory 2-9
Test Group Configuration (cf) Parameters Editor - command CF 2-5
Tick Timer Clear On Compare (VME2) - TMRF, TMRG 3-94
Tick Timer Increment (VME2) - TMRA, TMRB 3-89
Tick Timer No Clear On Compare (VME2) - TMRD, TMRE 3-92
Time Domain Reflectometry test (LANC) - TDR 3-120
Timer Accuracy Test (VME2) - TACU 3-87
Timer Clear on Compare test (MCC) - TMRnC 3-37
Timer Counters test (MCC) - TMRnA 3-33
Timer Free-Run test (MCC) - TMRnB 3-35
Timer Interrupts tests (MCC) - TMRnE 3-40

Timer Overflow Counter tests (MCC) - TMRnD 3-38
TMRA and TMRB tests (VME2) 3-89
TMRC test (VME2) 3-90
TMRD and TMRE tests (VME2) 3-92
TMRF and TMRG tests (VME2) 3-94
TMRH and TMRI tests (VME2) 3-96
TMRJ test (VME2) 3-98
TMRK test (VME2) 3-99
TMRnA tests (MCC) 3-33
TMRnB tests (MCC) 3-35
TMRnC tests (MCC) 3-37
TMRnD tests (MCC) 3-38
TMRnE tests (MCC) 3-40
toggle bits (RAM) 3-6
Translation Control register (MMU) 3-68

U
Used Page test (MMU) - USEDPAGE 3-72
USEDPAGE test (MMU) 3-72
utilities 2-4

V
VBR test (MCC) 3-42
Vector Base Register test (MCC) - VBR 3-42
Verify Default Tables (MMU) - TBLVERF 3-67
VME Interface Chip (VME2) tests 3-76
VME2 tests 3-76
REGA 3-77
REGB 3-79
SWIA 3-81
SWIB 3-83
SWIC 3-85
TACU 3-87
TMRA and TMRB 3-89
TMRC 3-90
TMRD and TMRE 3-92
TMRF and TMRG 3-94
TMRH and TMRI 3-96
TMRJ 3-98
TMRK test 3-99
VMEbus specification 1-11
VMEchip2 3-76

W
WALK test (MMU) 3-70
walking bit (VME2) 3-79
Watchdog Timer Board Fail (VME2) - TMRK 3-99
Watchdog Timer Board Fail test (MCC) - WDTMRB 3-44
Watchdog Timer Counter (VME2) - TMRJ 3-98
Watchdog Timer Counter test (MCC) - WDTMRA 3-43
Watchdog Timer Local Reset test (MCC) - TMRnB 3-46
WDTMRA test (MCC) 3-43
WDTMRB tests (MCC) 3-44
WDTMRC test (MCC) 3-46
word 1-12
WPPAGE test (MMU) 3-75
Write Protect Page test (MMU) - WPPAGE 3-75
write to IPIC registers - ACCESSB 3-141
write-read (RAM) 3-14
Writethrough test 3-61

Z
ZE command 2-10
Zero Pass Count - command ZP 2-11
ZP command 2-11

I
N
D
E
X