CIXCD

CIXCD Interface User Guide

Order Number EK-CIXCD-UG-003

digital equipment corporation
maynard, massachusetts

First Edition, May 1990
Second Edition, October 1990

This document was prepared and published by Educational Services Development and Publishing, Digital Equipment Corporation.

Contents

Installation Procedures

3 Site Preparation and Installation

4 Verification and Acceptance Testing

Examples

Figures

PAGE x INTENTIONALLY LEFT BLANK

About This Manual

This manual describes the CIXCD interconnect hardware option, which provides the parallel-to-serial interface between two corporate interconnect bus architecture protocols: the XMI bus and the dual-path CI bus. Included in the manual are the installation procedures, as well as the testing procedures to determine if the option is working correctly.

Intended Audience

This manual is intended for Digital Customer Service personnel, or customers who install and/or maintain this option. Such topics as installation and testing are covered in this manual. Digital personnel includes Customer Service branch-level engineers.

Manual Structure

This manual is divided into three main sections:

User information
Installation procedures
Service

The User Information section has two chapters: the CIXCD introduction that gives the option's specifications, and an overview of the VAX systems that use this option. The Installation Procedures section has two chapters: one on site preparation and installation and the other on verification and acceptance testing. The Service section has two chapters: a description of the CIXCD diagnostics, and a functional description of the CIXCD option. Also, there is an appendix with a description of all the visible registers.

Related Documents

The following manuals can provide additional information about the CIXCD option and the XMI bus.

Title	Order Number
CIXCD Technical Manual[1]	EK-CIXCD-TM
VAX 6000-400 System Technical User's Guide[1]	EK-640EA-TM
VAX 6000-400 Installation Guide	EK-640EA-IN
VAX 9000 Model 200 Installation Guide	EK-9200I-IN
VAX 9000 Model 200 Hardware User Guide	EK-9201U-UG
VAX Diagnostic User's Guide[1]	EK-VXDSU-UG
VAX Diagnostic User's Guide Update[1]	EK-VXDSU-U1
VAX Diagnostic System User's Guide[1]	EK-VX11D-UG
VAX Diagnostic Software Handbook[1]	AA-F152A-TE
SC008 Star Coupler User's Guide	EK-SC008-UG
CISCE-AA Installation Guide	EK-CISCE-UG
VAXcluster Service Reference Manual[1]	EK-VCSRM-PK

[1]Available only to Digital Customer Service personnel or licensed self-maintenance customers.

USER INFO

User Information

This section of the manual contains user information regarding the CIXCD. Important information is included regarding the various systems that use the CIXCD.

CHAPTER 1

1 Introduction

This chapter introduces the computer interconnect hardware interface options CIXCD-AA and CIXCD-AB. These options provide the interface between computer systems that use the high-speed XMI bus and the CI bus. Also included in this chapter are the physical description and specifications of the hardware.

The sections include:

- General description
- Specifications
- Physical hardware description

1.1 General Description

This section describes the components and the features of the CIXCD.

1.1.1 Components

The computer interconnect interface, shown in Figure 1–1, is designated the CIXCD port adapter. It is an intelligent interface, residing on a single module, that connects the XMI bus to the high-speed CI bus.

NOTE
All modules that plug into the XMI bus are considered XMI "native adapters." We will refer to the CIXCD as the CIXCD adapter, or as the port adapter.

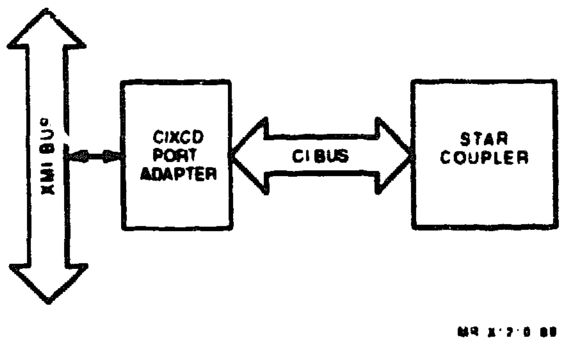


Figure 1–1 Simplified CIXCD Adapter Connection

As a buffered communications port, the CIXCD adapter completes high-level computer communications, thereby reducing software processing overhead. This is accomplished with hardware that provides all of the necessary data buffering, address translations, and serial data encoding/decoding. The CIXCD uses queue structures provided under the VMS operating system to transfer packet messages and to initiate the transfer of blocks of data between the VAX host memory system and/or other nodes within the VAXcluster configuration.

The CIXCD-AA adapter is made up of a single hardware module, T2080, a single header card assembly, 54-20225-01, and a single cable assembly: either 17-02894-01 for the CIXCD-AA or 17-02894-02 for the CIXCD-AB.

1.1.2 Features

- Resequencing dual path
- XMI bus design
- Typical performance of up to 8 Mbyte/s
- Parity on all internal buses and control stores
- Updateable control store
- Diagnostic loopback capability (both internal and external)
- Data integrity through cyclic redundancy checking (CRC)
- Round-robin arbitration at heavy loading, for each path
- Contention arbitration at light loading, for each path
- Packet-oriented data transmission
- Immediate acknowledgment of packet reception
- Operational modes:
  - Uninitialized
  - Disabled
  - Enabled

1.2 Specifications

CI General Specifications

Priority arbitration	
Light loading	Contention
Heavy loading	Round-robin
Parity	Cyclic redundancy check
Data format	Manchester-encoded serial packet

Environmental Specifications

Temperature	
Operating	10°C to 40°C (50°F to 104°F) ambient temperature with a gradient of 10°C (18°F)/hr
Storage/shipping	-40°C to 70°C (-40°F to 158°F) ambient temperature with a gradient of 20°C (36°F)/hr
Relative humidity	
Operating	10% to 90% with a maximum wet bulb temperature of 28°C (82°F) with a minimum dew point of 2°C (36°F) with no condensation
Storage/shipping	5% to 95% with no condensation
Altitude	
Operating	Sea level to 2.4 km (8000 ft) Maximum operating temperatures decrease by a factor of 1°C/1000 ft (1.8°F/1000 ft) for operation above sea level
Shipping/storage	Up to 9.1 km (30,000 ft) above sea level (actual or effective by means of cabin pressurization)
Shock	5 Gs peak at 7 to 13 ms duration in three axes mutually perpendicular (maximum)

Electrical Specifications

Power consumption:	Maximum ripple:
+5.0 V at 5.9 A nominal	300 mV
-5.2 V at 1.8 A nominal	150 mV
-2.0 V at 0.5 A nominal	150 mV

XMI Bus Specifications

Bus characteristics	
Type	Synchronous, pended
Width	64 data bits
Cycle time	64 ns
Priority arbitration	Modified round-robin
Parity	Even
Data transfers	Longword Quadword Octaword Hexword
Transmission characteristics	
Bandwidth	100 Mbytes/s (for HW transfers)
Length	17 inches
Bus loading (maximum)	16 nodes

CI Bus Specifications

Bus characteristics	
Width	Serial
External length (maximum)	45 m (147.64 ft)
Data transfer rate	140 Mbits/s (maximum)
Bus loading (maximum)	32 nodes
Cable type (BNCIA-XX)	Double-shielded coaxial
Cable impedance	50 ohms

1.3 Physical Hardware Description

Refer to Tables 1–1 and 1–2 for an overview of the CIXCD hardware components. The CIXCD-AA is used on VAX 9000 systems, while the CIXCD-AB is used on VAX 6000 systems.

Table 1–1 CIXCD-AA Hardware Components

Component	Part Number
CIXCD port adapter module	T2080
CIXCD header card assembly	54-20225-01
VAX 9000 bulkhead cable assembly	17-02894-01
Jumpers	12-14314-01
CIXCD Interface User Guide	EK-CIXCD-UG

Table 1–2 CIXCD-AB Hardware Components

Component	Part Number
CIXCD port adapter module	T2080
CIXCD header card assembly	54-20225-01
VAX 6000 bulkhead cable assembly	17-02894-02
Jumpers	12-14314-01
CIXCD Interface User Guide	EK-CIXCD-UG

1.3.1 CIXCD Module

The CIXCD module contains the following:

- XMI interface logic
- Packet buffer (PB) RAMs
- Control store RAMs
- Control store EEPROMs
- Local store RAMs
- Five gate arrays:
  - XMOV: data movers, XMI interface control
  - MCWI: PB memory control, CI wire interface and control

- MCDP: microprocessor, sequencer
- CIRT (2): Manchester encode/decode logic, byte framer, shift register

1.3.2 CIXCD Header Card Assembly

The CIXCD header card assembly is made up of two connected parts: a header card and a plastic cover. The header card is a circuit board that provides active logic to convert the received CI signals to ECL levels. The CI bulkhead cables connect to the header card, which then is plugged into the XMI backplane. The plastic cover provides dual protection: it protects the components from other cables plugged into the XMI backplane and it protects the other cables from the components. It also provides a key, so the assembly can only be plugged into the backplane the correct way. Figure 1–2 shows the CIXCD header card assembly.

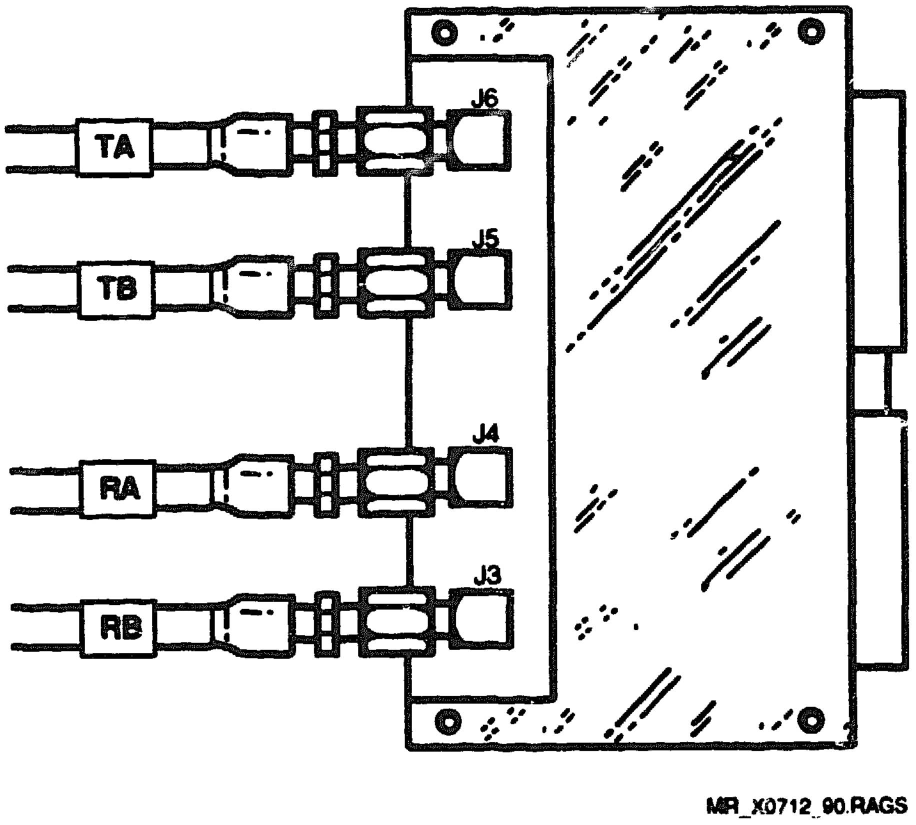


Figure 1–2 CIXCD Header Card Assembly

1.3.3 CI Bulkhead Cable Assembly

The CIXCD is connected from the XMI backplane, through the header card and the CI bulkhead cable to the CI bulkhead connector panel. The CI bulkhead cable is made up of four coaxial cables, each of which connects to the header card on one end and to the CI bulkhead connector panel on the other end. The CI bulkhead connector panel is mounted on the I/O panels of the cabinet (quad panel size for VAX 6000 systems) and provides an EMI/RFI shield without compromising signal integrity. Figures 1-3 and 1-4 show the CI bulkhead cable.

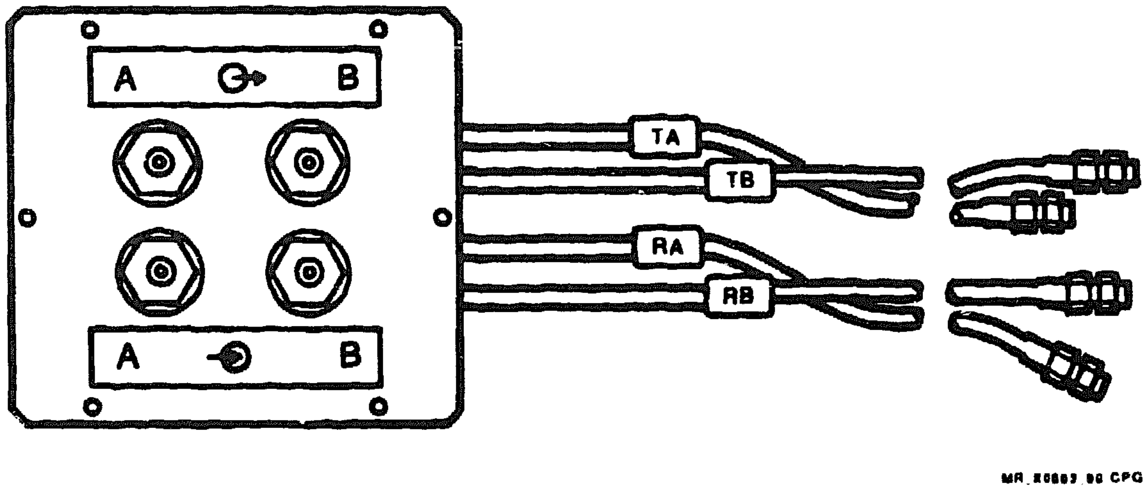


Figure 1-3 CIXCD-AA Bulkhead Cable

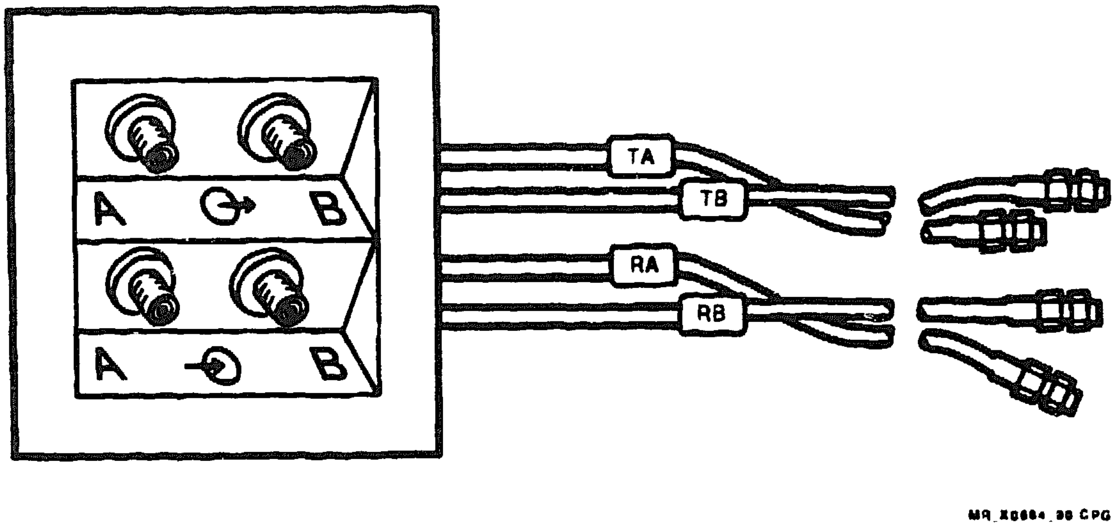


Figure 1-4 CIXCD-AB Bulkhead Cable

1.3.4 Jumpers

The jumpers are required to provide specific configuration information to the MCDP microprocessor. They are placed in zones D2 and E2 of the XMI backplane (refer to Section 3.5 for placement information).

CHAPTER 2

2 Systems Overview

This chapter describes the families of hardware systems that can use the CIXCD interface. All these systems have the XMI bus as either the system bus or as an I/O bus.

They include:

- VAX 6000 family (model 200, model 300, model 400, model 500)
- VAX 9000 family (model 200, model 400)

2.1 VAX 6000 Family

All members of the VAX 6000 family can use the CIXCD-AB interface. In this family there are four entries, each of which comes with many models. Table 2–1 shows the members of this family.

Table 2–1 VAX 6000 Family

Model	Members	Model	Members
VAX 6000 model 200	Model 210	**VAX 6000 model 400**	Model 410
	Model 220		Model 420
	Model 230		Model 430
	Model 240		Model 440
			Model 450
			Model 460
VAX 6000 model 300	Model 310	**VAX 6000 model 500**	Model 510
	Model 320		Model 520
	Model 330		Model 530
	Model 340		Model 540
	Model 350		Model 550
	Model 360		Model 560

2.1.1 VAX 6000 Cabinets

All VAX 6000 systems use the same standard system cabinet. Figure 2–1 shows the VAX 6000 family cabinet.

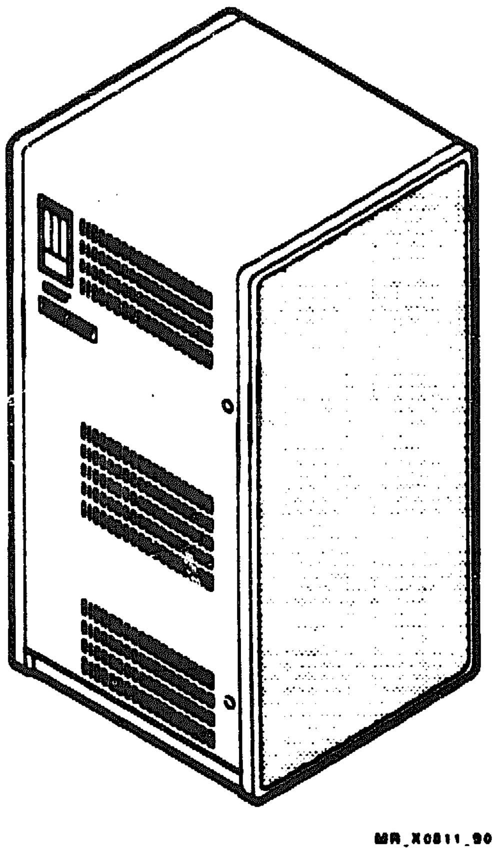


Figure 2-1 VAX 6000 Cabinet

2.2 VAX 9000 Family

All members of the VAX 9000 family can use the CIXCD-AA interface. In this family there are two entries, one of which comes in many models. Table 2–2 shows the members of this family.

Table 2–2 VAX 9000 Family

Model	Members
VAX 9000 model 200	Model 210
VAX 9000 model 400	Model 410
	Model 420
	Model 430
	Model 440

2.2.1 VAX 9000 Cabinets

The VAX 9000 family of systems can have many different cabinet configurations, depending on model type. Model 210 has a unique set of cabinets, while model 400 systems all use the same cabinet types, but with different configurations. Figures 2-2 and 2-3 show the different cabinets.

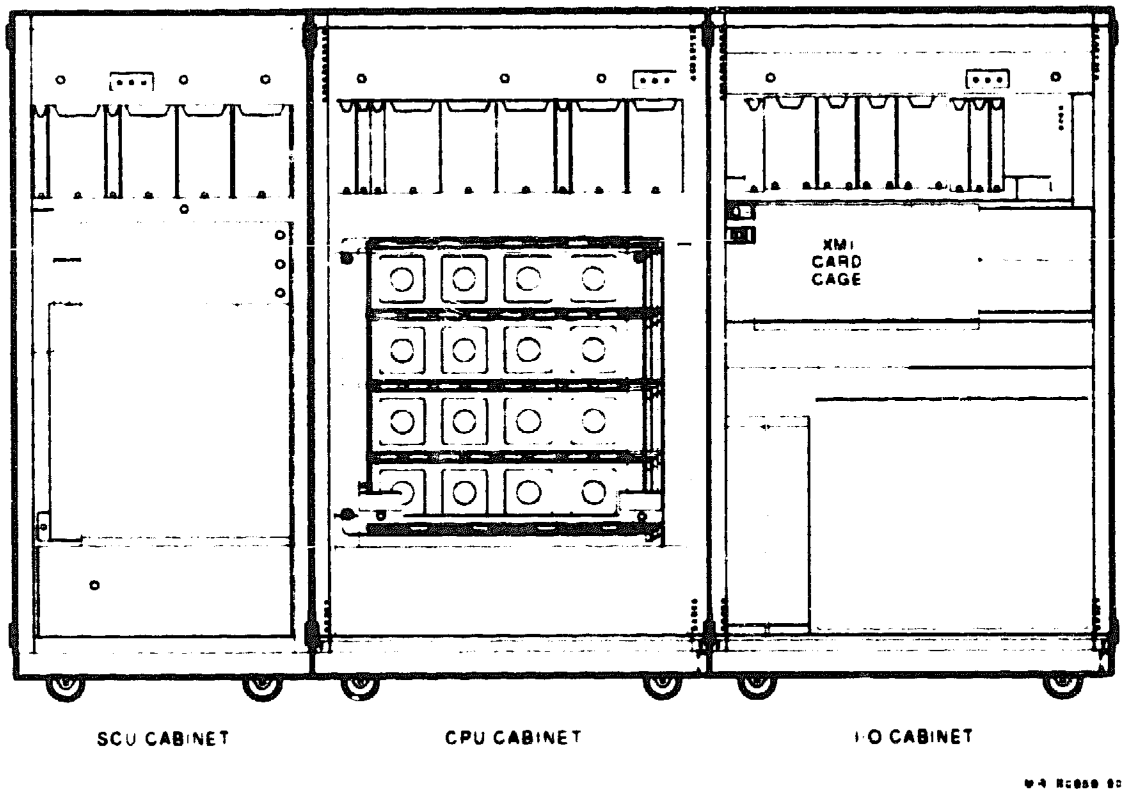


Figure 2-2 VAX 9000 Model 210 Cabinet

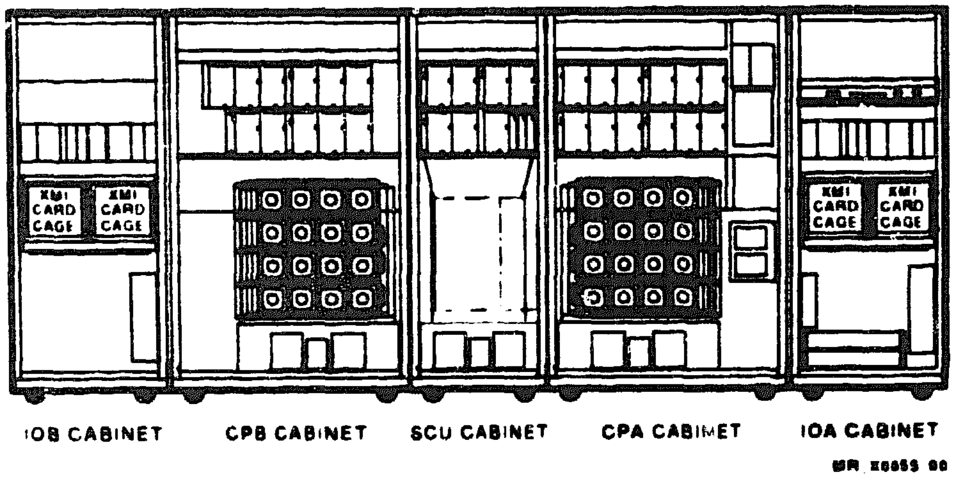


Figure 2–3 VAX 9000 Model 400 Cabinets

INST

PROC

Installation Procedures

This section of the manual contains the information necessary to install the CIXCD option.

CHAPTER 3

3 Site Preparation and Installation

This chapter describes the site preparations necessary for installation, and steps that must be followed to install the CIXCD option.

The sections include:

- Operating environment
- System configurations
- Unpacking and inventory
- Installation and configuration
- Backplane jumpers and configuration selection

3.1 Operating Environment

This section describes the system physical specifications that must be available to install the CIXCD option.

3.1.1 Physical Elements

The CIXCD option requires one I/O slot in the XMI backplane.[1]

There must be the equivalent of two available I/O connector panel openings in the appropriate cabinet[2] to hold the CI bulkhead cable connector panel. For VAX 6000 systems, there must be one 102 mm × 102 mm (4 in × 4 in) opening or two adjacent 51 mm × 102 mm (2 in × 4 in) openings. For VAX 9000 systems, there must be one available I/O connector panel opening (4.2 in square).

[1] All XMI slots are I/O slots, except slot 7 in VAX 9000 systems, slots 6-9 in VAX 6000 mod[illegible] 500 systems, and slots 5-A in all other VAX 6000 systems.

[2] CPU cabi[illegible] for VAX 6000 systems, I/O cabinet for VAX 9000 systems.

3.1.2 System Environment and Grounding Elements

Consult the appropriate system installation manual for information regarding system environment and grounding requirements.

3.2 System Configurations

The CIXCD option may be installed in many systems under many different circumstances. For VAX 9000 systems, it might be a new installation with the CIXCD option already installed, or it might be an existing standalone system being upgraded to add a CIXCD. For VAX 6000 systems, you might have an existing unclustered system being added to a cluster, or you might have a clustered system being upgraded from a CIBCA option to the CIXCD option.

For clustered VAX 6000 and VAX 9000 systems performing initial installation, since the module and hardware are already installed, go to Section 3.5.

For nonclustered VAX 9000 systems being upgraded to a clustered system, begin the installation at Section 3.4.2.

For nonclustered VAX 6000 systems being upgraded to a clustered system, go to Section 3.4.2 to start the installation.

For clustered VAX 6000 systems, with the CIBCA option, being upgraded to a CIXCD option, start the installation at Section 3.4.1.

3.2.1 VAXcluster Revision Levels

Ensure that the CIXCD hardware and microcode revision levels are consistent with the revision level of the cluster, which must be at least at the N1 level. Consult the VAXcluster Revision Matrix Document (K-RM-CLUSTER-V-0) for the latest information on N1. For the minimum acceptable revision levels, refer to Section C.1.

3.3 Unpacking and Inventory

The customer is responsible for the actual movement of the equipment to the installation site. Ensure that all equipment for the CIXCD option is moved to the designated installation site.

3.3.1 Unpacking the Shipping Boxes

1. Locate and open the box marked "OPEN ME FIRST". It contains the shipping/accessory list.
2. Notify the customer of any opened boxes or cartons, and document this fact on the installation report.
3. Check all boxes for external damage (dents, holes, or crushed corners).
4. Notify the customer of all damage, and list all damage on the installation report.
5. Open all remaining boxes.

3.3.2 Inventory

1. Check the shipment against the shipping/accessory list.
2. Inspect the equipment for damage. Note any damage on the installation report.
3. If damage is extensive, notify the Customer Service unit manager for instructions on how to proceed.
4. Notify the Customer Service unit manager of any missing or incorrect items.
5. Request that the customer contact the shipping carrier to locate any missing items.
6. Request that the Customer Service unit manager check with the Digital Equipment Corporation Traffic and Shipping Department if the shipping carrier does not have the missing items.

3.4 Installation and Configuration

There are three separate procedures involved in the installation of the CIXCD option. Installations require one, two or all three of the following procedures. Perform them in the order listed.

1. Removal of the CIBCA option. This is required only on currently clustered systems that are upgrading to the CIXCD option. Refer to Section 3.4.1.
2. Installation of the CIXCD option. This is performed on all installations. Refer to Section 3.4.2.

3. Updating the VAX 6000 console microcode. This is required on any VAX 6000 systems with microcode that does not recognize the CIXCD. Refer to Section 3.4.3.

3.4.1 Removal of the CIBCA Option

This section describes the steps involved in removing the CIBCA option from VAX 6000 systems.

CAUTION
Use an antistatic wrist ground strap when you work on a VAXBI system with its covers removed or when you handle any VAXBI module. Do not remove any module from the card cage until you have antistatic packaging ready for it.

1. Power down the system.
2. Open the front door of the cabinet and ground yourself with the attached wrist strap.
3. Open the Plexiglas doors to gain access to the modules.
4. Remove the T1045 module and the adjacent T1046 module from the card cage, taking care to place each one in an antistatic package. Note the slot numbers.
5. Open the rear door of the cabinet and ground yourself with the attached wrist strap.
6. Checking for the correct slots, carefully remove the 51 mm (2 in) cable (the outermost one, PN 17-01504-01) from section C of the backplane. Then remove the inner cable (PN 17-01504-02). Refer to Figure 3–1.
7. Remove the internal CI bulkhead cables and the dummy connectors from sections D and E of the T1046 module slot. Refer to Figure 3–2.

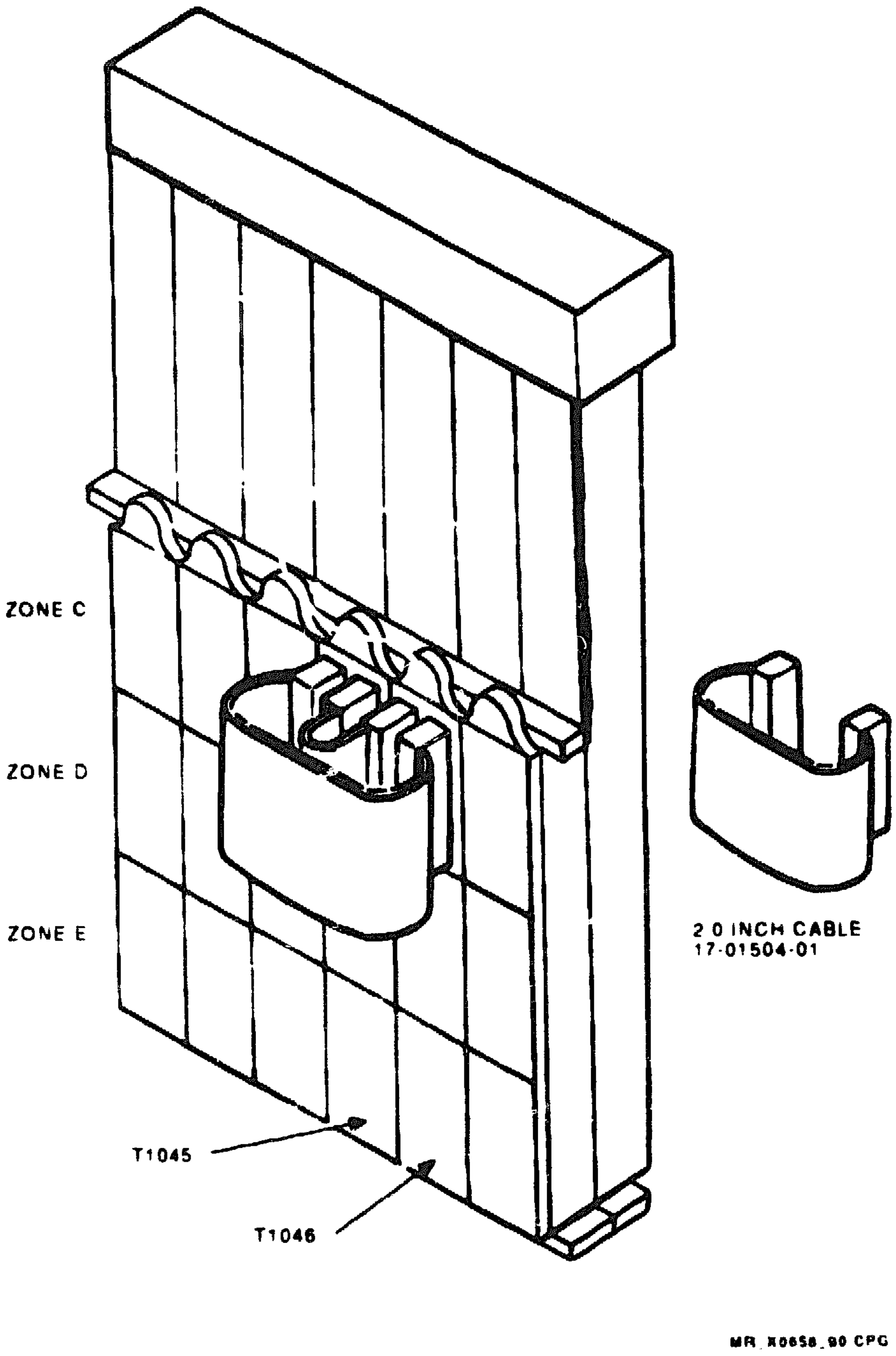


Figure 3–1 CIBCA Intermodule Cables

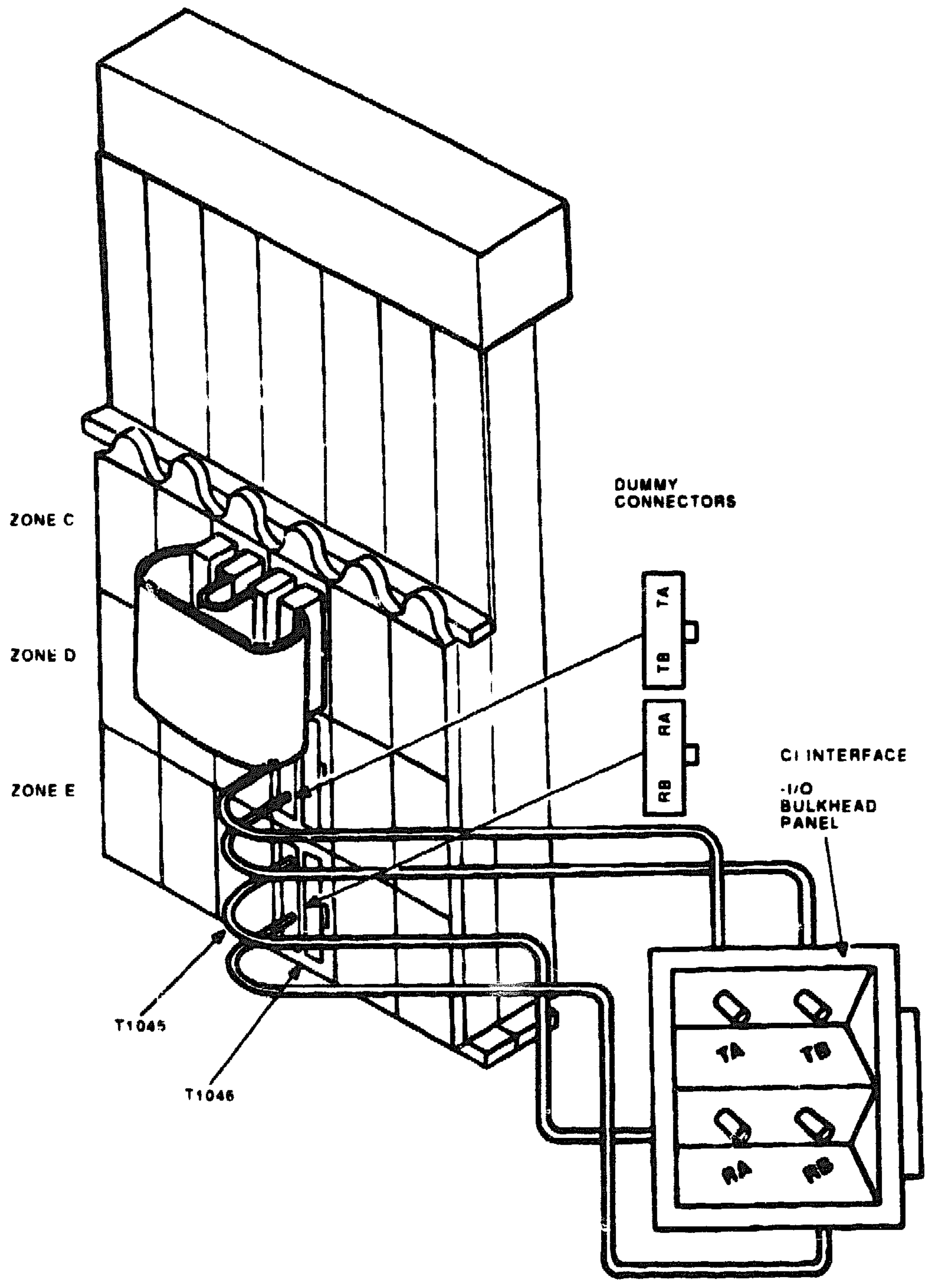


Figure 3–2 CIBCA Cable Location

8. Remove the node ID plug from section A of the T1045 module slot. Refer to Figure 3–3.

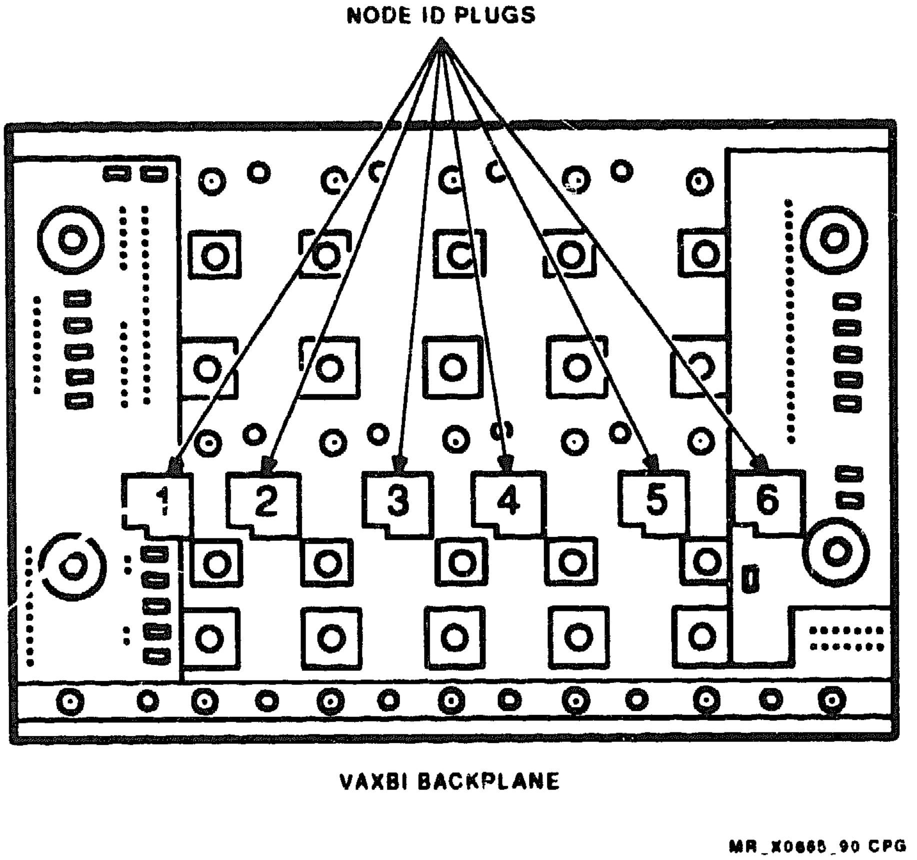
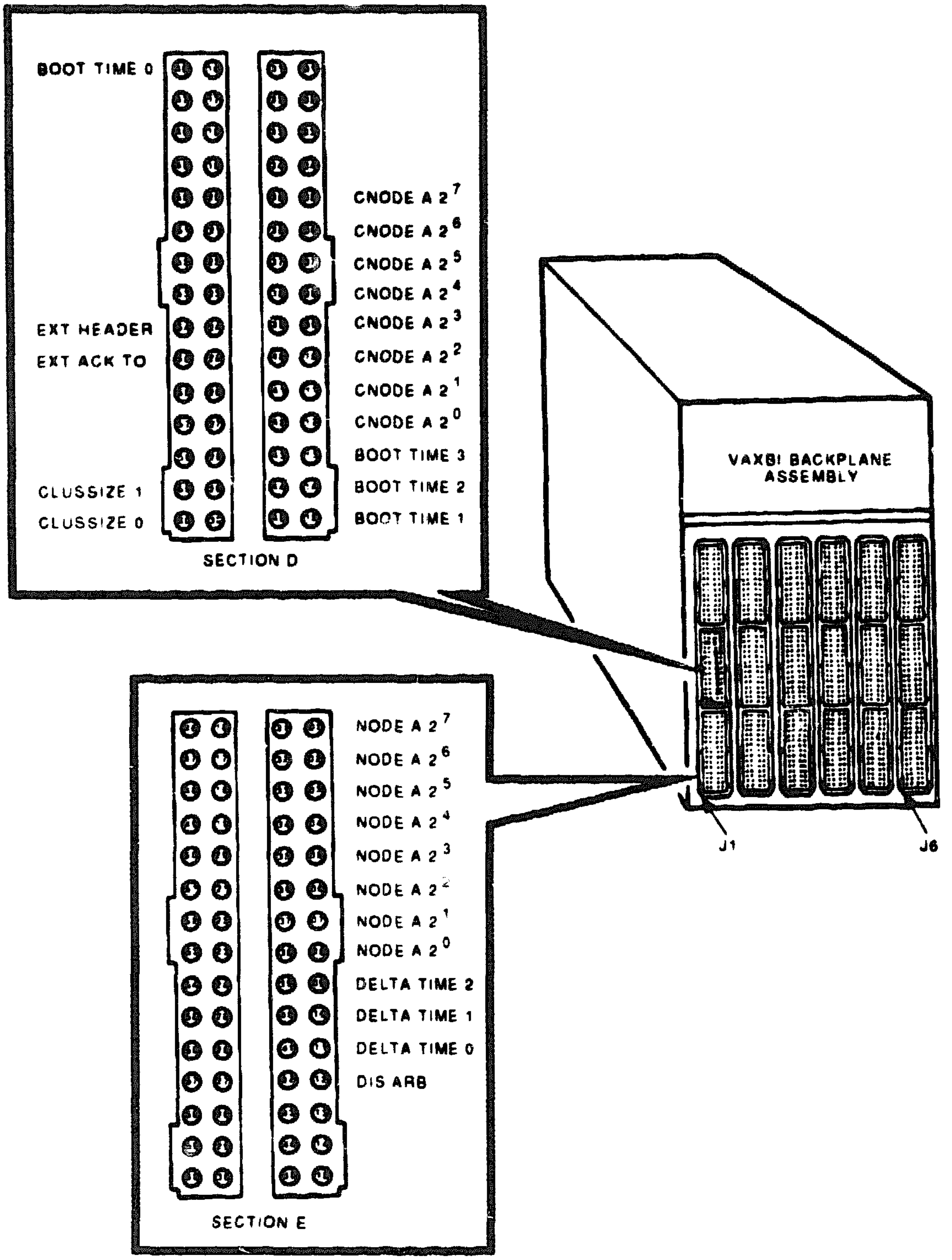


Figure 3–3 Node ID Plug Location

9. Remove all jumpers from sections D and E of the same slot. Refer to Figure 3–4.



MR_X0660 80 CPG

Figure 3–4 CIBCA Jumpers

3.4.2 Installation of the CIXCD Option

This section describes the steps required to install and connect the T2080-00 module (CIXCD option) to VAX 9000 systems.

CAUTION
You must use an antistatic wrist strap when you work on any VAX system with its covers removed or when you handle any XMI module. Do not remove any module from its antistatic packaging until you are ready to install it.

1. Power down the system.
2. Open the front door of the cabinet and ground yourself with the attached wrist strap.
3. Open the Plexiglas doors to access the XMI card cage.
4. Remove the T2080-00 module from the antistatic package and carefully insert it into an unoccupied module slot in the XMI card cage (I/O slots only).

   On VAX 6000 systems, you cannot configure the CIXCD in the center six slots (5-A) of the XMI card cage (the restriction for model 500 systems is slots 6-9). This area of the card cage is covered by the arbitration daughter card, so the header card and CIXCD configuration jumpers cannot be installed.

   On the VAX 9000 systems, this restriction applies only to slot 7. All others are I/O slots.

   Figure 3–5 shows the T2C80-00 module installed in the XMI card cage. Notice the proper module orientation in relation to side 1 of the module. Side 1 of the T2080-00 module has the greater number of components on it, and should be facing to the right, when viewing the module from the front of the cabinet.

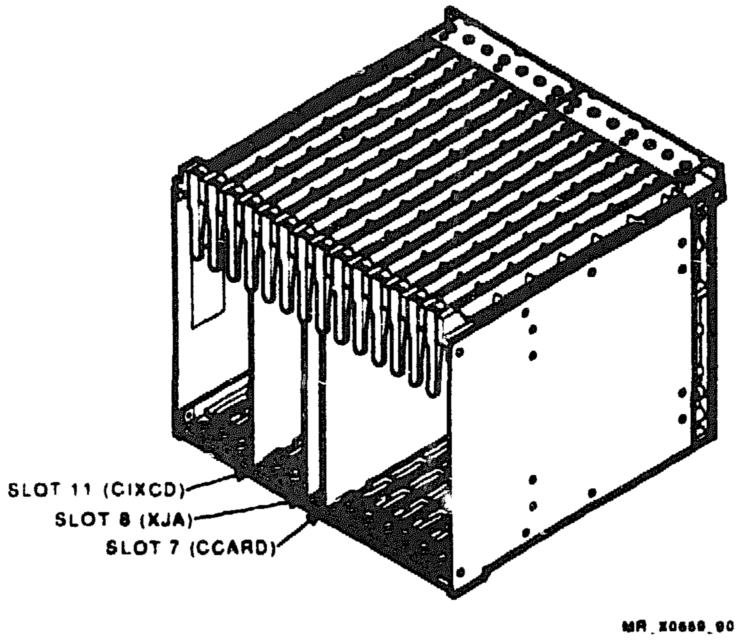


Figure 3–5 CIXCD Module Installed in Card Cage

5. Close the Plexiglas doors and remove the wrist strap. Open the rear door of the cabinet and ground yourself with the attached wrist strap.
6. Install the configuration jumpers into the I/O header in zones D2 and E2 of the slot containing the T2080-00 module (refer to Figure 3–7).
7. Route the internal CIXCD bulkhead cables and install the I/O bulkhead panel.
8. Using a torque wrench (PN 29-27973-01), connect each of the CIXCD bulkhead cables to its corresponding coaxial connector on the header card (refer to Figure 1–2). The torque specification for this connector is 7-10 inch-pounds. The cables and their corresponding connectors are:

   TA - J6
   TB = J5
   RA = J4
   RB = J3

9. Carefully insert the header card into the I/O headers in zones D1 and E1 of the slot containing the T2080-00 module. Side 1 of the card (with the cables attached) should be facing to the right as you view the backplane from the rear of the cabinet. The header cover is keyed, so the header card cannot be plugged in incorrectly. This also secures the header card to the backplane.

   NOTES

   1. **Unlike the CIBCA adapter for the VAXBI bus, the CIXCD does not use "dummy" connectors.**
   2. **The header cover is keyed in such a way that the header card can only fit in D1/E1.**
   3. **The recommended quiet slot time for all nodes in clusters is 10. A quiet slot time of 10 is *required* for all clusters that contain a CIXCD. Refer to Section C.2.**

3.4.3 Updating the VAX 6000 Console Microcode

Some VAX 6000 systems must have updated console microcode to boot through the CIXCD. This section describes the steps required to determine if the update is needed and to perform the update.

3.4.3.1 Determining the Need for Updated Microcode

At the >>> prompt, issue the SHOW CONFIGURATION command. If the system does not recognize the CIXCD, then it needs updated microcode. Refer to Examples 3–1 and 3–2.

NOTE
The XMI device type code for the CIXCD is 0C05.

```
>>> SHOW CONFIG

     Type              Rev
1+   KA64A    (8001)   8002
2+   MS62A    (4001)   0002
4+   ???????  (0C05)   0621
D+   DWMBA/A  (2001)   0002
E+   DWMBA/A  (2001)   0002

XBI  D
1+   DWMBA/B  (2107)   0007
5+   DMB32    (0109)   210B
6+   DEBNA    (410F)   0248

XBI  E
1+   DWMBA/B  (2107)   0007
2+   KDB50    (010E)   0F1C
6+   TBK50    (410E)   0248

>>>
```

Example 3–1 SHOW CONFIGURATION With Old Microcode

```
>>> SHOW CONFIG

     Type              Rev
1+   KA64A    (8001)   8002
2+   MS62A    (4001)   0002
4+   CIXCD    (0C05)   0621
D+   DWMBA/A  (2001)   0002
E+   DWMBA/A  (2001)   0002

XBI  D
1+   DWMBA/B  (2107)   0007
5+   DMB32    (0109)   210B
6+   DEBNA    (410F)   0248

XBI  E
1+   DWMBA/B  (2107)   0007
2+   KDB50    (010E)   0F1C
6+   TBK50    (410E)   0248

>>>
```

Example 3–2 SHOW CONFIGURATION With New Microcode

3.4.3.2 Performing the Update

1. Locate and insert the update tape cartridge into the TK tape drive following these steps:
   a. Check that the tape drive LOAD/UNLOAD button is out and the power is on.
   b. Open the drive door by pushing the cartridge-release handle to the left.
   c. Hold the tape cartridge with the write-protect switch up, the arrow on the side of the cartridge pointing toward the tape drive.
   d. Insert the cartridge into the TK drive.
   e. Close the door by firmly moving the cartridge-release handle to the right.
   f. Depress the LOAD/UNLOAD button.
2. Turn the lower keyswitch to the Update position. Refer to Figure 3–6.

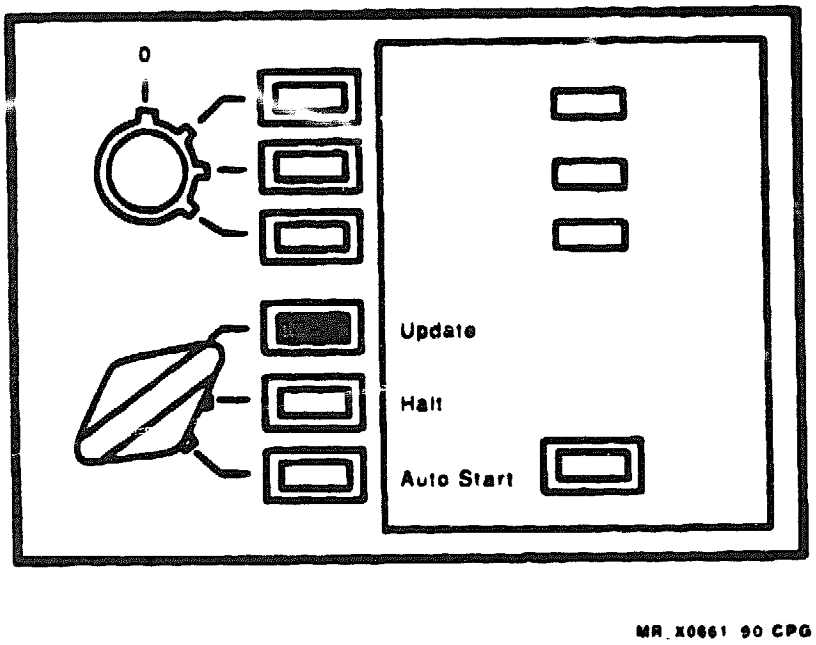


Figure 3–6 The Update Position

3. At the >>> prompt, type the PATCH EEPROM command, and reply "Y" to the PROCEED? query.

```
>>>
>>> PATCH EE
?6D EEPROM Revision = 2.0/4.4

?6F Tape image Revision = 2.0/4.E

Proceed with EEPROM update? (Y or N) >>> Y
?69 EEPROM changed successfully.

>>>
>>>
```

4. If the VAX 6000 system is a multiprocessor system, the new console microcode must be propagated to all other processors. At the >>> prompt, type in the UPDATE ALL command.
5. Check that the new microcode has been correctly loaded into the system. At the >>> prompt, type the INITIALIZE command. Check the microcode revision level listed in the last line of the printout.
6. The VAX 6000 is now ready to boot through the CIXCD.

```
>>>
>>> PATCH EE
?6D EEPROM Revision = 2.0/4.4

?6F Tape image Revision = 2.0/4.E

Proceed with EEPROM update? (Y or N) >>> Y
?69 EEPROM changed successfully.
>>>
>>> I
F   E   D   C   B   A   9   8   7   6   5   4   3   2   1   0   NODE #
    A   .   A   .   .   .   .   .   .   P   P   A   M   P       TYP
    O   .   +   .   .   .   .   .   .   +   +   +   +   +       STF
    .   .   .   .   .   .   .   .   .   E   E   .   .   B       BPD
    .   .   .   .   .   .   .   .   .   +   +   .   .   +       ETF
    .   .   .   .   .   .   .   .   .   E   E   .   .   B       BPD

.   .   .   .   .   .   .   .   .   +   .   .   .   +   +   .   XBI E +

    .   .   .   .   .   .   .   .   .   .   .   .  A1   .       ILV
    .   .   .   .   .   .   .   .   .   .   .   .  32   .       32 Mb

ROM = 3.1  EEPROM = 2.0/4.E  SN = AG84102189

?2D For Secondary Processor 4
?53 EEPROM revision mismatch.  Secondary processor has revision 2.0/4.4

?2D For Secondary Processor 5
?53 EEPROM revision mismatch.  Secondary processor has revision 2.0/4.4

>>>
>>>
>>> U ALL
>>>
>>> I
F   E   D   C   B   A   9   8   7   6   5   4   3   2   1   0   NODE #
    A   .   A   .   .   .   .   .   .   P   P   A   M   P       TYP
    O   .   +   .   .   .   .   .   .   +   +   +   +   +       STF
    .   .   .   .   .   .   .   .   .   E   E   .   .   B       BPD
    .   .   .   .   .   .   .   .   .   +   +   .   .   +       ETF
    .   .   .   .   .   .   .   .   .   E   E   .   .   B       BPD

.   .   .   .   .   .   .   .   .   +   .   .   .   +   +   .   XBI E +

    .   .   .   .   .   .   .   .   .   .   .   .  A1   .       ILV
    .   .   .   .   .   .   .   .   .   .   .   .  32   .       32 Mb

ROM = 3.1  EEPROM = 2.0/4.E  SN = AG84102189

>>>
>>>
```

Example 3–3 Updating VAX 6000 Console Microcode

3.5 Backplane Jumpers and Configuration Selection

To correctly configure the CIXCD, place the jumpers in the correct place in the backplane. The XMI bus backplane-pin numbering is the same as the VAXBI bus backplane-pin numbering. There are 29 possible pins that may need jumpers on the backplane, in sections D and E of the backplane. The corresponding jumpers are denoted W1 through W30, with W9 being reserved. Refer to Figure 3–7 to determine which jumper corresponds to which backplane pin. Note that the jumpers are only placed in zones D2 and E2.

The functions that can be modified by jumper placement are the following:

- CI node address
- Boot time
- Disable arbitration
- Extend header
- Alter delta time (quiet slot time)
- Cluster size
- Extend ACK timeout

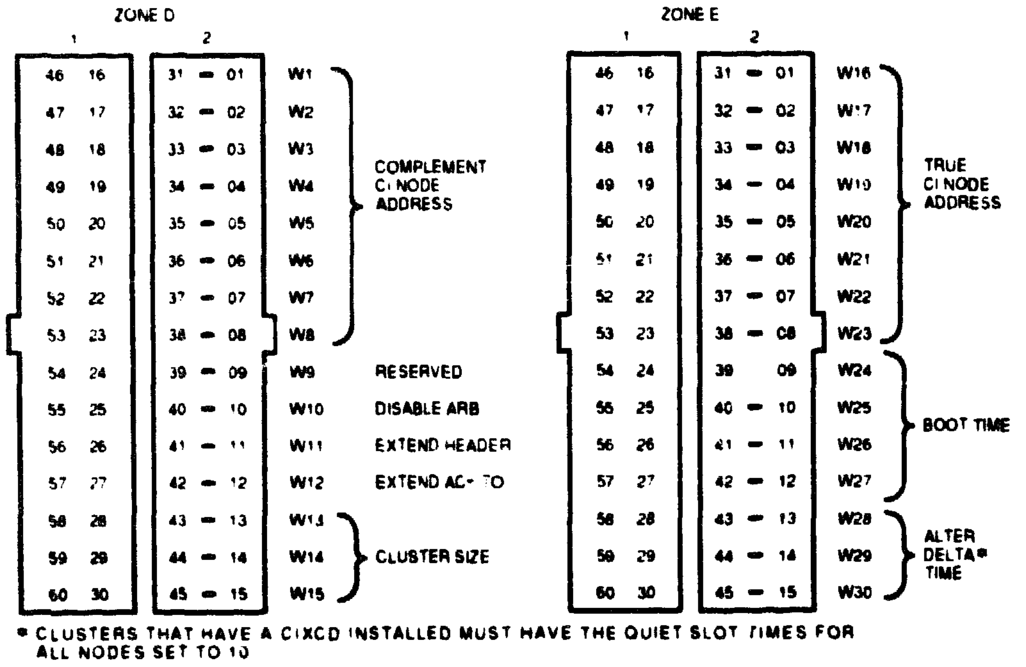


Figure 3–7 CIXCD Jumper Pinning

3.5.1 CI Node Address

The CI node address is obtained from the CIXCD port adapter module's backplane slot, with both the CI node address and its complement configured exactly the same. To configure the jumpers for the node's address, refer to Table 3–1 for the true-address and the complement-address configurations.

Table 3–1 CI Node Address Table — True/Complement

	True Address							
CI Node Addr(10)	W16 E1/31	W17 E2/32	W18 E3/33	W19 E4/34	W20 E5/35	W21 E6/36	W22 E7/37	W23 E8/38
0	Out	Out	Out	Out	Out	Out	Out	Out
1	Out	Out	Out	Out	Out	Out	Out	In
2	Out	Out	Out	Out	Out	Out	In	Out
.				.				
.				.				
.				.				
223	In	In	Out	In	In	In	In	In

	Complement Address							
CI Node Addr(10)	W1 D1/31	W2 D2/32	W3 D3/33	W4 D4/34	D5 D5/35	D6 D6/36	D7 D7/37	D8 D8/38
0	Out	Out	Out	Out	Out	Out	Out	Out
1	Out	Out	Out	Out	Out	Out	Out	In
2	Out	Out	Out	Out	Out	Out	In	Out
.				.				
.				.				
.				.				
223	In	In	Out	In	In	In	In	In

NOTES
CI node addresses 224 through 255 are reserved for Digital Equipment Corporation.

The address in the complement address field must be exactly the same as the address in the true address field.

3.5.2 Boot Time

The boot time is the length of time the port will wait after power-up to exit the uninit state. Refer to Table 3–2 for the correct jumper configuration. All jumpers out is the default configuration.

NOTE
The *boot time* is the maximum length of time the port will wait after power-up to exit the uninit state. In normal operation, the port will exit this state in conjunction with the virtual machine bootstrap (VMB) long before exhaustion of this timer. To ensure proper operation, all jumpers must be left in the default (out) configuration.

Table 3–2 Boot Time Backplane Jumpers

	W24		W25	W26	W27
Time	E9	39	E10/40	E11/41	E12/42
500	Out	Out	Out	Out	(Default)
400	Out	Out	Out	In	
300	Out	Out	In	Out	
200	Out	Out	In	In	
100	Out	In	Out	Out	
000	Out	In	Out	In	
900	Out	In	In	Out	
800	Out	In	In	In	
700	In	Out	Out	Out	
600	In	Out	Out	In	
500	In	Out	In	Out	
400	In	Out	In	In	
0300	In	In	Out	Out	
0200	In	In	Out	In	
0100	In	In	In	Out	
0000	In	In	In	In	

3.5.3 Disable Arbitration

The disable arbitration bit, when set, defeats the normal arbitration sequence and allows the CI controller logic to transmit after waiting only one basic quiet slot (delta) time. The jumper that controls this bit is W10 (D10/40).

Jumper out	=	Allow normal CI arbitration (default)
Jumper in	=	Disable normal CI arbitration

3.5.4 Extend Header

Jumper W11 (D11/41) controls the extend header bit that, when set, allows the CI controller logic to extend the number of bit sync characters in the header.

Jumper out	=	Normal header (default)
jumper in	=	Extended header

3.5.5 Alter Delta Time

These bits force the CI controller logic to increase the basic quiet slot delta time. Refer to Table 3–3 for the configurations.

Table 3–3 Quiet Slot Time Backplane Jumpers

Quiet Slot Count	W28 E13/43	W29 E14/44	W30 E15/45	
7	Out	Out	Out	
10	Out	Out	In	(Required setting for CIXCD)
Reserved	Out	In	Out	
Reserved	Out	In	In	
Reserved	In	Out	Out	
Reserved	In	Out	In	
Reserved	In	In	Out	
Programmable	In	In	In	

NOTE
For clusters that have a CIXCD installed, *all* nodes in that cluster must have their quiet slot time set for 10. Refer to Section C.2.

3.5.6 Cluster Size

The cluster size bits cause the arbitration logic to arbitrate for more than 16 nodes (which is the default). If any node in the cluster has a node number greater than 15, the appropriate cluster size must be indicated with these jumpers. Refer to Table 3–4 for the configurations.

Table 3–4 Cluster Size Backplane Jumpers

Node Count	W13 D13/43	W14 D14/44	W15 D15/45	
16	Out	Out	Out	(Default)
32	Out	Out	In	
64	Out	In	Out	
128	Out	In	In	
224	In	Out	Out	
Reserved	In	Out	In	
Reserved	In	In	Out	
Reserved	In	In	In	

3.5.7 Extend ACK Timeout

The extend ACK timeout bit forces the CI controller logic to increase the timeout period for an ACK return. The jumper that represents this bit is W12 (D12/42).

Jumper out	=	Short timeout (default)
Jumper in	=	Long timeout

NOTE
A system with no jumpers would be configured in the following way:

- **CI node address = 0**
- **Boot time = 1500 s**
- **Normal CI arbitration**
- **Normal header**
- **Quiet slot delta time = 7**
- **Cluster size = 16**
- **Short ACK timeout**

CHAPTER 4

4 Verification and Acceptance Testing

This chapter describes the procedures to verify that the system is in good working condition. The sections include:

- Diagnostic verification
- System maintenance tools

4.1 Diagnostic Verification

After completing the CIXCD installation, verify that the system works as it is supposed to. This section lists the steps required to test out the system.

4.1.1 Diagnostic Programs

There is a specific set of diagnostic programs used to determine if the CIXCD adapter is working properly. Figure 4–1 describes the testing procedure for the CIXCD, while Table 4–1 lists the diagnostic programs that must be run as part of this procedure.

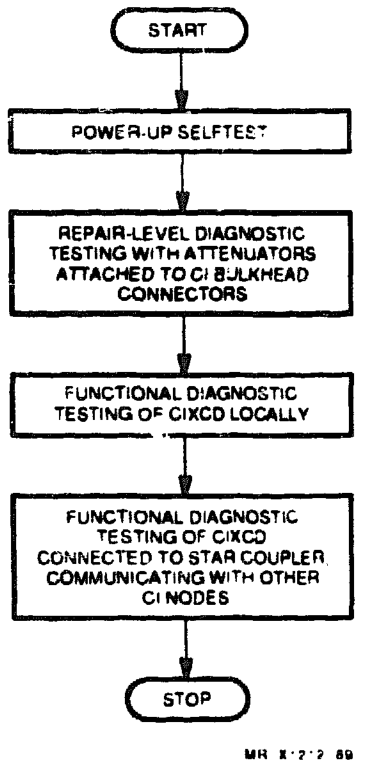

START
POWER-UP SELFTEST
REPAIR-LEVEL DIAGNOSTIC TESTING WITH ATTENUATORS ATTACHED TO CI BULKHEAD CONNECTORS
FUNCTIONAL DIAGNOSTIC TESTING OF CIXCD LOCALLY
FUNCTIONAL DIAGNOSTIC TESTING OF CIXCD CONNECTED TO STAR COUPLER COMMUNICATING WITH OTHER CI NODES
STOP
MR X-2-2 89

Figure 4–1 CIXCD Acceptance Testing Flow Diagram

Table 4–1 CIXCD Diagnostic Programs

Program Designation	Program Level	Program Title
XCDST	5	CIXCD self-test
EVGEA	3	CIXCD repair-level diagnostic
EVGEB	3	CIXCD microcode update utility
EVGAA	3	CI functional diagnostic 1
EVGAB	3	CI functional diagnostic 2
EVGAC	3	Cluster functional diagnostic
EVXCI	2R	CI exerciser diagnostic

4.1.2 Power-Up Self-Test

After a system power-up or reset, the CIXCD runs its self-test (XCDST), which provides logic level testing for the module. It is designed to provide coverage of > 95% of all possible "stuck at" faults.

After successfully completing XCDST, the CIXCD turns on its yellow self-test LED. The pass/fail information is also available to the console terminal. To get the steps necessary to retrieve this information, refer to the *VAX 9000 Model 200 Hardware User Guide*.

On VAX 6000 systems, there is console printout at power-up that shows the status of the system elements, including the CIXCD. Example 4–1 shows a console printout in which the CIXCD is at node C.

```
F   E   D   C   B   A   9   8   7   6   5   4   3   2   1   0      NODE #

    A   A   A   .   .   M   M   M   M   .   P   P   P   P          TYP
    o   o   +   .   .   +   +   +   +   .   +   +   +   +          STF
    .   .   .   .   .   .   .   .   .   .   E   E   D   B          BPD
    .   .   .   .   .   .   .   .   .   .   +   +   +   +          ETF
    .   .   .   .   .   .   .   .   .   .   E   E   D   B          BPD

.   .   .   .   .   .   .   .   .   .   +   .   +   +   -   +      XBI D +
.   .   .   .   .   .   .   .   .   .   +   +   .   +   +   +      XBI E +

    .   .   .   .   .   A4  A3  A2  A1  .   .   .   .   .          ILV
    .   .   .   .   .   32  32  32  32  .   .   .   .   .          128MB
```

Example 4-1 VAX 6000 Console Printout

In this system, the CIXCD is node C and it passed its self-test. This is shown by the plus sign (+) in the third row (labeled on the right side with "STF"). We can tell that nodes D and E are not CIXCDs because they have an "o" in the self-test row, which indicates that they did not run a self-test, and the DWMBA/A is the only option that does not run self-test. The notation for failing self-test is the minus sign (-). For more information about this printout, refer to the *VAX 6000-400 System Technical User's Guide* (EK-640EB-TM).

4.1.3 Preliminary Diagnostic Setup

Prior to running the macrodiagnostics, you must set up the system both hardware- and software-wise. The hardware setup requires the installation of CI bus loopback connectors; the software setup requires the running of the VAX Diagnostic Supervisor (VDS) program.

4.1.3.1 Loopback Connectors

Before running the diagnostics, make the following CI bus loopback connections on the CI bulkhead connector panel located at the back of the panel (refer to Figure 4-2).

1. Using one of the attenuator pads (PN 12-19907-01) and two of the modularity cables (PN 70-18530-00), connect the transmit A to receive A.

2. Perform the same connection for path B using another attenuator pad and two more modularity cables, connecting transmit B to receive B.

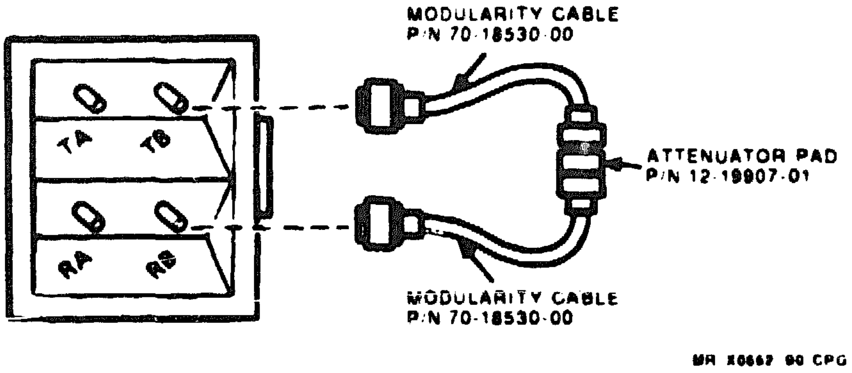


Figure 4–2 Diagnostic Loopback Cable Connections

4.1.3.2 Loading the VAX Diagnostic Supervisor (VDS) Program

The level 3 diagnostics are stand-alone programs that require the support of VDS in order to run. Follow the steps listed below:

1. Load VDS into physical memory from the console load device. This procedure can vary, depending on the type of system in which the CIXCD is being installed. (Refer to the applicable system installation manual for the VDS load and run procedures.)

2. Identify the CIXCD adapter to VDS, specifying its node configuration parameters. The ATTACH/SELECT sequence of VDS varies according to the system.

   NOTE
   Before installing the CIXCD hardware, you should be familiar with the ATTACH and SELECT sequences of VDS for the processor that is used.

   If VDS is loaded through a CIXCD, VDS assigns the designation PAA0 to that CIXCD. Other CIXCDs to be tested should be assigned other designations (for example, PAB0 or PAC0). Be careful to avoid the command SELECT ALL under these circumstances, as it will cause PAA0 to be tested. (Not booting through the CIXCD allows you to use both the PAA0 designation and the SELECT ALL command.)

   VAX 6000 example:

```
DS>  ATTACH CIXCD HUB PAB0 C 5
```

VAX 9000 example:

```
DS>  ATTACH XJA HUB XJA0 0 8
DS>  ATTACH CIXCD XJA0 PAB0 3 7
```

3. **Select the CIXCD adapter as the test unit, as follows:**

```
DS>  SELECT PAB0, XJA0
```

4. **Show the unit selected, as follows:**

```
DS>  SHOW SELECT
```

4.1.4 Repair-Level Testing

The repair-level diagnostic for the CIXCD is EVGEA. You must successfully complete five passes of this program to satisfy acceptance testing requirements.

1. Ensure that the diagnostics are accessible through the default load path during diagnostic acceptance testing. This might require changing diagnostic media in the current load-path device.
2. Load EVGEA, as follows:

```
DS>  LOAD EVGEA
```

3. Set the desired VDS control flags:
   - Enable printing of the number and title of each test before it runs.
   - Enable halting on a detected error.

```
DS>  SET FLAGS TRACE,HALT
```

4. Start the diagnostic program.

```
DS>  START/PASS:5
```

Example 4-2 provides trace printouts for EVGEA.

```
DS> st/pass:1

.. Program: EVGEA CIXCD Functional Level 3 Diag , revision 2 0, 25 tests,
   at 13:53:14.27.
Testing: _PAB0

Test - 1 Scan Data Path Verification
  Subtest - 1 Scan Data Register Loopback
  Subtest - 2 Port Scan Control Register
  Subtest - 3 Port Scan Shift Register
  Subtest - 4 PMCS RAM Data Path

Test - 2 PMCS EEPROM region Checksum
  Subtest - 1 Backup region Functional Microcode Checksum
  Subtest - 2 Primary region Functional Microcode Checksum
  Subtest - 3 Backup region Self-test Microcode Checksum
  Subtest - 4 Primary region Self-test Microcode Checksum

Test - 3 PMCS RAM Memory

Test - 5 CIXCD Node RESET
  Subtest - 1 Run Self-test via XBER NRST - Check XBER status
  Subtest - 2 XMOV loads RAM from EEPROM - Check XMOV and RAM data

Test - 6 Scan Visibility Bus CS Address Field

  Loading Microcode file EVGEA.BIN

Test - 7 XMI Device (XDEV) and Port Serial Number Register (PSNR)

Test - 8 ROM Based Diagnostic Interface
  Subtest - 1 RBD 0 - Power Up Self-test
  Subtest - 2 RBD 2 Test 1 - Port Local Store Data Integrity
  Subtest - 3 RBD 2 Test 2 - Port Local Store Address Independence
  Subtest - 4 RBD 2 Test 3 - Port Local Store Literal Addressing
  Subtest - 5 RBD 2 Test 4 - Port Local Store TCB Base Relative Addressing
  Subtest - 6 RBD 3 Test 1 - Port Packet Buffer Data Integrity
  Subtest - 7 RBD 3 Test 2 - Port Packet Buffer Address Independence
  Subtest - 8 RBD 4 - XMI Commander
  Subtest - 9 RBD 1 - Verify CI jumpers
  Subtest - 10 RBD 1 - External Loopback on Path A
  Subtest - 11 RBD 1 - External Loopback on Path B

.. End of run, 0 errors detected, pass count is 1,
   time is 30-MAR-1990 13:54:13.34

DS>
```

Example 4-2 Trace Printout for EVGEA (One Pass)

4.1.5 CI Functional Level Testing

With the CI bus cables and attenuator pads providing signal loopback, load and run the CI functional diagnostics EVGAA and EVGAB. You must successfully complete a minimum of five passes of each diagnostic program to satisfy acceptance testing requirements. Examples 4-3 and 4-4 show trace printouts for EVGAA and EVGAB respectively.

1. Ensure that the diagnostics are accessible through the default load path during diagnostic acceptance testing. This might require changing diagnostic media in the current load-path device.
2. Load the EVGAA diagnostic program.

   ```
   DS>  LOAD EVGAA
   ```

3. Set the desired VDS control flags:
   a. Enable printing of the number and title of each test before it runs.
   b. Enable halting on a detected error.

   ```
   DS>  SET FLAGS TRACE,HALT
   ```

4. Start the EVGAA diagnostic program.

   ```
   DS>  START/PASS:5
   ```

5. After five successful passes of EVGAA, load and run EVGAB.

   ```
   DS>  LOAD EVGAB
   DS>  START/PASS:5
   ```

6. Disconnect the attenuators from the ends of the CI bus cables in preparation for routing and connecting the cables to the star coupler.
7. Route and connect the cables to the star coupler.

   NOTE
   For information on connecting the coaxial CI bus cables, refer to the *SC008 Star Coupler User's Guide*.

8. Repeat steps 2, 4, and 5 of this procedure.

```
DS> st/pass:1

.. Program: EVGAA - CI Functional Part I, Level 3, revision 6.0, 17 tests
   at 11:58:12.80.

Testing _PAB0

Event Flag 1 SET = Load CI Microcode
Event Flag 2 SET = Print Queue Entries
Event Flag 3 SET = REQID Loop Function in Test 1

Testing Device _PAB0

EEPROM Revision = 0020   Functional Revision = 00FC

Test 1:  Cluster Configuration

                Cluster Configuration for Path A
                ********************************
You CANNOT Differentiate between a CI780, CI750, or a CIBCI remotely.
(PS = Path Select,  TP = Transmit Path,  RP = Receive Path)

 Node     Device      Hard  Soft    Extended Port     Path      P T R
Number     Type       Rev.  Rev.    Functionality    Status     S P P
------   --------   ------------   -------------   --------   - - -
  02      CIXCD     0001  0001      0FFF8100(X)       OK       A A A
  0D      HSC50        0001         00000000(X)       OK       A A A

                Cluster Configuration for Path B
                ********************************
You CANNOT Differentiate between a CI780, CI750, or a CIBCI remotely.
(PS = Path Select,  TP = Transmit Path,  RP = Receive Path)

 Node     Device      Hard  Soft    Extended Port     Path      P T R
Number     Type       Rev.  Rev.    Functionality    Status     S P P
------   --------   ------------   -------------   --------   - - -
  02      CIXCD     0001  0001      0FFF8100(X)       OK       B B B
  0D      HSC50        0001         00000000(X)       OK       B B B

                 Nodes NOT Listed do not exist on Cluster

Test 2:  SETCKT with Various Masks and M Values
Test 3:  SETCKT for Each Valid Port
Test 4:  SETCKT for Invalid Port
Test 5:  REQID Basic
Test 6:  REQID With 6 Packets on DGFQ
Test 7:  Datagram Discard
Test 8:  Response Queue Available Interrupt
Test 9:  Send Datagram
Test 10:  SNDMSG With No Virtual Circuit Open
Test 11:  Send Message Crossing Page Boundary
Test 12:  Message Length Test
Test 13:  Packet Size Violation
Test 14:  Send Loopback (SNDLB)
Test 15:  SNDLB Full Buffer on Path A
Test 16:  SNDLB Full Buffer on Path B
Test 17:  SNDLB Automatic Path Selection
.. End of run, 0 errors detected,  pass count is 1,
   time is 02-APR-1990 11:58:36.40
DS>
```

Example 4–3   Trace Printout for EVGAA (One Pass)

```
DS> st/pass:1

.. Program: EVGAB - CI Functional Part II, Level 3, revision 6.0, 12 tests
   at 12:05:24.83

Testing _PAB0

Event Flag 1 SET = Load CI Microcode
Event Flag 2 SET = Print Queue Entries
Event Flag 3 SET = Not Used By This Diagnostic


Testing Device _PAB0

EEPROM Revision = 0020   Functional Revision = 00FC

Test 1:  Send Data with Offset Combinations
Test 2:  Request Data with Offset Combinations
Test 3:  Invalidate Translation Cache
Test 4:  SNDMDAT in Enabled/Maintenance State
         Port functionality does not support this test

Test 5:  SNDMDAT in Enabled State
         Port functionality does not support this test

Test 6:  REQMDAT in Enabled/Maintenance State
         Port functionality does not support this test

Test 7:  REQMDAT in Enabled State
         Port functionality does not support this test

Test 8:  SEND RESET in Enabled State
Test 9:  Queue Protocol
Test 10:  Buffer Read Access
Test 11:  Buffer Write Access
Test 12:  Write to Global Buffer
.. End of run,  0 errors detected,  pass count is 1.
   time is 02-APR-1990 12:05:47.34
DS>
```

Example 4-4 Trace Printout for EVGAB (One Pass)

4.1.6 System Functional Level Testing

With the CIXCD still connected to the star coupler, load and run the system functional diagnostic EVGAC. You must successfully complete a minimum of five passes of this diagnostic program to satisfy acceptance testing requirements.

1. Ensure that the diagnostics are accessible through the default load path during diagnostic acceptance testing. This might require changing diagnostic media in the current load-path device.
2. Load the EVGAC diagnostic program.

   ```
   DS>  LOAD EVGAC
   ```

3. **Set the desired VDS control flags:**
   a. **Enable printing of the number and title of each test before it runs.**
   b. **Enable halting on a detected error.**

```
DS>  SET FLAGS TRACE,HALT
```

4. **Start the EVGAC diagnostic program.**

```
DS>  START/PASS:5
```

Example 4-5 shows trace printouts for EVGAC.

```
DS> st/pass:1

.. Program: EVGAC - CI Functional Exerciser, revision 1.1, 8 tests
   at 11:51:09.89.

Testing _PAB0

Test 1:  Local Configuration
Test 2:  Local Adapter Test
Test 3:  Datagram Test
Test 4:  Virtual Circuits Test
Test 5:  Message Test
Test 6:  Multiple Message Test
Test 7:  Write/Read Buffer Test
Test 8:  Activity Test
.. End of run,  0 errors detected,  pass count is 1,
   time is 02-APR-1990 11:55:53.84
DS>
```

Example 4-5 Trace Printout for EVGAC (One Pass)

4.2 System Maintenance Tools

In addition to the specified stand-alone diagnostics, there are some additional tools available to verify the CIXCD adapter. Table 4–2 provides a summary of the VAXcluster system maintenance tools.

Table 4–2 VAXcluster System Maintenance Tools

Tool	Function
EVXCI	A level 2R multipurpose exerciser that provides local CI interface functional testing, as well as a means to determine the ability of VAXcluster nodes to reliably communicate using the CI bus.
ERF	The Errorlog Report Formatter. The user may create reports with the system errors catalogued in various ways.[1]
VAXsim	A VAX System Integrity Monitor utility program that monitors and filters errors as they are logged by the VMS operating system. The VAXsim program provides the user with a warning mechanism that quickly identifies an option that has either failed or has degraded operationally.[2]
SHOW CLUSTER	Allows the display of a large variety of utility information relevant to the configuration and operation of the VAXcluster system of which the host system is a member.[3]
SET HOST/HSC	Allows a terminal on a host VMS system to effectively become an HSC50/70 terminal. The user can then issue any standard HSC50/70 commands and look at or control the HSC50/70 just as if it were a terminal connected directly to one of the HSC50/70 terminal ports.[4]

[1]For more information, consult the *VAX/VMS Error Log Utility Reference Manual*, (AA-Z402C-TE).

[2]For more information, consult the *VAX System Integrity Monitor User Guide*, (AD-KN80A-TI).

[3]For more information, consult the *VAX/VMS Show Cluster Utility Manual*, (AA-LA46A-TE).

[4]For more information, consult the *VAX/VMS DCL Dictionary*, (AA-LA12A-TE) under SET HOST/HSC.

SERVICE

Service

This section of the manual contains service information regarding the CIXCD. It includes information about the self-tests, as well as information about the diagnostics, both ROM-based (RBDs) and macros. It also details the acceptance tests and procedures required as part of installation.

CHAPTER 5

5 Diagnostics

This chapter describes the diagnostic programs that are used to test the CIXCD. The main sections of this chapter are:

- CIXCD self-test/ROM-based diagnostics
- Macrodiagnostics

5.1 CIXCD Self-Test/ROM-Based Diagnostics

The CIXCD self-test (XCDST) and the ROM-based diagnostic (RBDs) are microdiagnostics for the CIXCD.

5.1.1 XCDST

The CIXCD self-test is run automatically on a system power-up or XMI reset. It can also be run as RBD0, from the RBD user interface (refer to Section 5.1.2.1). The test, located in the CIXCD EEPROM, is designed to provide logic-level testing for the module, making sure that the CIXCD is fully operational. The testing is done under the control of the CIXCD port processor, using a bottom-up approach, meaning that each test executes only after all preceding tests have successfully completed.

When XCDST runs automatically and fails, the indication is that the self-test passed (STP) LED on the module does not turn on, and the self-test failed (STF) bit 10 in the XMI bus error register (XBER) is set. In addition, an error code is written into the port diagnostic control/status register (PDCSR). The error code is the number of the failing test.

NOTE
The diagnostic tests are numbered in decimal (1–22), while the error codes are in hexadecimal (1–16). Be aware that, for example, error code 11 indicates a failure of test 17.

There are additional tests, performed by the functional microcode, whose failure prevents the STP LED from turning on. In the course of loading itself into the control store RAMs, the functional microcode must test the CI jumpers on the backplane to make sure they contain the correct information. If an error is found, one of the codes listed in Table 5–1 is put into the PDFLT field of the PDCSR. Note that bit <07> is always set by these codes.

If no errors are found by either the diagnostics or the functional microcode, the code A0(hex) is put into the PDFLT field.

Table 5–1 Functional Microcode Test Failure Error Codes

Error Code	Description
81	Illegal cluster size value in jumpers.
82	Node number does not validate. The node number given in the backplane jumpers is greater than the cluster size given in the jumpers.
83	Quiet slot time is illegal. The slot time field in the backplane jumpers is an illegal value.
84	Node and complement node addresses different. The node number and complement node number address fields in the jumpers do not match.

5.1.1.1 XCDST Tests

The tests that are performed by the XCDST are as follows:

Test 1: Port Processor ALU Status and Branch Test

Subtest 1: ALU Z bit set, ALU NZ bit cleared
Subtest 2: ALU NZ bit set, ALU Z bit cleared
Subtest 3: ALU N bit
Subtest 4: ALU C bit
Subtest 5: LOOP COUNT ZERO condition
Subtest 6: PDP RREG
Subtest 7: PDP YREG <07:00>
Subtest 8: PDP YREG <07:08>
Subtest 9: PDP micro status register
Subtest 10: Masked branch bits <03:01>
Subtest 11: Masked branch bits <03:02>
Subtest 12: Masked branch bits <03>

Test 2: ALU Arithmetic/Logical Function Test

Subtest 1: ALU ADD function
Subtest 2: ALU SUB function
Subtest 3: ALU AND function
Subtest 4: ALU OR function
Subtest 5: ALU XOR function

Test 3: General Purpose Register Test

Subtest 1: GPR data independence
Subtest 2: GPR address independence

Test 4: Microsequencer Stack Testing

Test 5: Internal Bus Loopback

Subtest 1: PORT IB buffer through the X bypass and Y bypass
Subtest 2: PORT IB buffer through the X MUX and Y MUX
Subtest 3: PORT IB direct through the X MUX and Y MUX

Test 6: Interval Timer Testing

Subtest 1: Verify short interval
Subtest 2: Verify long interval

Test 7: Local Store Testing

Subtest 1: Local store data independence
Subtest 2: Local store address independence
Subtest 3: Local store literal addressing
Subtest 4: Local store TCB base relative addressing

Test 8: Memory Control and Wire Interface Tests

Subtest 1: Mover A packet buffer address register
Subtest 2: Mover B packet buffer address register
Subtest 3: Port packet buffer address register
Subtest 4: Memory controller register independence
Subtest 5: Packet buffer data independence
Subtest 6: Packet buffer address independence

Test 9: Data Mover A

Subtest 1: Data mover A byte count register (MVABCR)
Subtest 2: Data mover A XMI address register (MVAADR)
Subtest 3: Data mover A XMI next page register (MVANPR)
Subtest 4: Data mover A register independence

Test 10: Data Mover B

Subtest 1: Data mover B byte count register (MVBBCR)
Subtest 2: Data mover B XMI address register (MVBADR)
Subtest 3: Data mover B XMI next page register (MVBNPR)
Subtest 4: Data mover B register independence

Test 11: XMI Commander test

Subtest 1: Commander address A register (CMDRAAR)
Subtest 2: Commander address B register (CMDRABR)
Subtest 3: Commander XMI data1 LO register (CDAT1LR)
Subtest 4: Commander XMI data1 HI register (CDAT1HR)
Subtest 5: Commander XMI data2 LO register (CDAT2LR)
Subtest 6: Commander XMI data2 HI register (CDAT2HR)
Subtest 7: Commander register independence
Subtest 8: Commander longword data transfer

Test 12: XMI Responder

Subtest 1: Responder data register
Subtest 2: Responder offset

Test 13: Data Mover Loopback

Subtest 1: Hexword data transfer
Subtest 2-16: Mover byte rotation
Subtest 17: Page data transfer
Subtest 18: Page boundary crossing
Subtest 19: Next page empty interrupt

Test 14: XMI Bus Error Register test

Subtest 1: Commander TTO, NRR, and RIDNAK
Subtest 2: Mover A TTO and CNAK
Subtest 3: Mover B TTO and CNAK
Subtest 4: Interrupt TTO and CNAK
Subtest 5: XBER command NAK and NRR
Subtest 6: XBER no read response
Subtest 7: XBER read error response bit
Subtest 8: XBER read sequence error bit
Subtest 9: Commander write data NAK
Subtest 10: Mover B write data NAK
Subtest 11: XBER parity error
Subtest 12: XBER node halt

Test 15: XMI Device Register (XDEV)

Test 16: XMI Failing Address Registers

Subtest 1: XFADR data test
Subtest 2: XFAER data test

Test 17: Port Processor Internal Conditions

Subtest 1: Stack overflow
Subtest 2: Stack underflow
Subtest 3: Control store parity error
Subtest 4: X MUX parity error
Subtest 5: Y MUX parity error
Subtest 6: Internal bus parity error
Subtest 7: AC_LO

Test 18: MCWI Error Detection Logic

Subtest 1: MCWI PORT_IB_H receive parity checker
Subtest 2: MCDP packet buffer read parity checker
Subtest 3: MCWI PB_OR_IB_DATA_H parity checker
Subtest 4: Mover A packet buffer write parity checker
Subtest 5: Mover B packet buffer read parity error

Test 19: XMOV Error Detection Logic

Subtest 1: Data mover A register PB_OR_IB parity error
Subtest 2: Data mover B register PB_OR_IB parity error
Subtest 3: Commander register PB_OR_IB parity error
Subtest 4: Interrupt controller register PB_OR_IB parity error
Subtest 5: Responder register PB_OR_IB parity error
Subtest 6: Mover B packet buffer PB_OR_IB parity error
Subtest 7: Mover A byte parity error
Subtest 8: Mover B byte parity error

Test 20: Interrupt Control Registers

Subtest 1: Interrupt vector/interrupt priority level register
Subtest 2: Interrupt destination mask register
Subtest 3: Software initiated interrupt
Subtest 4: Send XMI error interrupt
Subtest 5: Send write error interrupt

Test 21: CI Internal Maintenance Loopback

Subtest 1: Verify CI jumpers
Subtest 2: Loopback at CIC, path A
Subtest 3: Loopback at CIC, path B
Subtest 4: Internal loopback path A
Subtest 5: Internal loopback path B
Subtest 6: Internal loopback 4K packet
Subtest 7: Swap true/complement node address, path A
Subtest 8: Swap true/complement node address, path B
Subtest 9: Internal loopback bad CRC, path A
Subtest 10: Internal loopback bad CRC, path B
Subtest 11: Internal loopback RX byte parity error, path A
Subtest 12: Internal loopback RX byte parity error, path B
Subtest 13: Internal loopback TX byte parity error, path A
Subtest 14: Internal loopback TX byte parity error, path B
Subtest 15: Internal loopback invalid tail pointer, path A
Subtest 16: Internal loopback invalid tail pointer, path B

Test 22: MCDP Priority Interrupt Logic

Subtest 1: Interrupt level disable
Subtest 2: Interrupt level 7
Subtest 3: Interrupt level 6
Subtest 4: Interrupt level 5
Subtest 5: Interrupt level 4
Subtest 6: Interrupt level 3
Subtest 7: Interrupt level 2
Subtest 8: Interrupt level 1
Subtest 9: Interrupt level 0
Subtest 10: Interrupt priority encoder

5.1.2 RBDs

In addition to the tests that run automatically, there are additional tests stored in the CIXCD EEPROM space. These are the RBDs that can be run by calling up the RBD user interface.

Once you have called up the RBD user interface, you can run the RBDs individually, getting pass/fail information from each test. RBD 0 is the XCDST, while RBD 1 through 3 are additional tests that can be run. The diagnostics are:

RBD 0 — Power-Up Self-Test (XCDST)

RBD 1 — CI External Maintenance Loopback[1]

Test 1: Verify CI jumpers
Test 2: External loopback on path A
Test 3: External loopback on path B

RBD 2 — Port Local Store Exerciser

Test 1: Local store data integrity
Test 2: Local store address independence
Test 3: Local store literal addressing
Test 4: Local store TCB base relative addressing

RBD 3 — Port Packet Buffer Exerciser

Test 1: Packet buffer data integrity
Test 2: Packet buffer address independence

[1] Loopback connectors are required. Refer to Section 4.1.3.1.

5.1.2.1 RBD User Interface

The RBD user interface communicates with the console of the host device through the XMI communications register (XCOMM). It is made up of two main parts: the RBD parser, and the RBD commands. You can get to the RBD user interface by "Z"ing to the CIXCD node that you want to test.

For VAX 9000 systems, type:

```
>>>
>>>Z  1A                 ;1 is the XMI number,
                         ;A is the XMI node number of the CIXCD

[  Use Ctrl/P to exit Z-MODE  ]

                         ;Note that there is no prompt.
```

For VAX 6000 systems, type:

```
>>>
>>>Z  C                 ;C is the XMI node number of the CIXCD

?43  Z connection successfully started
C>>
```

Then type the console command language (CCL) command TEST/RBD (or use the abbreviation T/R) and the interface returns with the RBD prompt: "RBDn> ", where n represents XMI node number of the CIXCD under test.

RBD Parser

The RBD parser supports the minimum subset of commands that are required by the XMI RBD specification. It accepts either uppercase or lowercase command input, converting all input to uppercase before acting on it. The parser performs the command if it recognizes correct syntax, or it returns the bell character and a question mark if it sees incorrect syntax.

RBD Commands

The RBD parser supports three commands (START, QUIT, and EXAMINE), four control characters, and eight qualifiers to the START command. Only the correct abbreviations of the commands and qualifiers will be accepted by the parser. Table 5-2 describes the commands, Table 5-3 describes the qualifiers for START, and Table 5-4 explains the control characters.

Table 5–2 RBD Commands

Command	Acceptable Format	Description
START n	ST n	This command starts the specified diagnostic (n).
QUIT	QU	This command returns control to the CIXCD's functional firmware. After this command is issued, the module will be initialized and the command T/R must be issued to resume RBD execution.
EXAMINE x	E x	This command examines the contents at address x. X must be a hexadecimal number.

Table 5–3 START Qualifiers

Qualifier	Name	Description
/LE	Loop on test	This qualifier causes the diagnostic to loop on the test where the first error is detected, with error reporting still in effect (if enabled). The loop can be terminated only by typing a Ctrl/C, Ctrl/Z, or Ctrl/Y. Upon interruption of the tests (by typing a control character), an error summary is printed on the console terminal. Refer to Section 5.1.2.2.
/HE	Halt on error	This qualifier causes the diagnostic to stop execution after the first error is detected. It reports the error and executes the cleanup code. CONTINUE ON ERROR is the default condition.
/IS	Inhibit summary	This qualifier causes the diagnostic to suppress the printing of the summary message on the console at the completion of the selected RBD.
/IE	Inhibit error output	This qualifier causes the diagnostic to suppress the printing of detected error reports on the console terminal. This qualifier is useful in combination with /LE. Error reporting is enabled by default. Refer to Section 5.1.2.2.

Table 5–3 (Cont.) START Qualifiers

Qualifier	Name	Description
/TR	Enable trace test	This qualifier enables the printing of the number of each test before it is executed. This condition is disabled by default.
/BE	Bell on error	This qualifier causes the diagnostic to send a bell character to the terminal whenever an error is encountered. This is useful when the error printout is inhibited and the diagnostic is looping on an intermittent error.
/P=n	Pass count	This qualifier allows the diagnostic to run multiple passes of each test (*n* is a decimal number). The default is one pass. If n = 0, the diagnostic makes an infinite number of passes, halted only by typing Ctrl/C, Ctrl/Z, or Ctrl/Y on the console.
/T=n[:m]	Test number	This qualifier allows the user to invoke an individual test (/T = n) or a range of tests (/T = n:m) (both *n* and *m* are decimal numbers). The default is all tests.

Table 5–4 RBD Control Characters

Character	Function
Ctrl/U	Running: Ignored.
	Parser: Disregard previous input.
Ctrl/Z	Running: Stop execution of diagnostic and execute cleanup code.
	Parser: Same as QUIT command.
Ctrl/C	Running: Same as Ctrl/Z.
	PARSER: Same as Ctrl/U.
Ctrl/Y	Running: Stop execution of diagnostic and do *not* execute cleanup code.
	Parser: Same as Ctrl/U.

5.1.2.2 RBD Information Printout

When you run an RBD, it immediately prints a header line, and at the end of the test (either successful or unsuccessful), it prints a summary report. Additional information is also printed, if necessary. A trace message is printed if the trace qualifier is used, and an error report is supplied if the test detects an error.

Header Line Format

When a diagnostic is started with the *ST n* command, the first line printed is a header line that consists of the test name followed by the diagnostic revision number. The revision number is of the form *R.uu*, where R is the major revision level and uu is the minor (or update) revision level. This line is only printed once and cannot be inhibited. An example of the format follows:

Example:

```
;XCD_ST      1.00
```

Trace Message Format

If the trace qualifier /TR is appended to the command string, the trace messages are printed in the following format:

Example:

```
; T01  T02  T03  T04  T05  T06  T07  T08  T09  T10
; T11  T12
```

At the beginning of each test, the test number about to be executed is sent to the console. It is preceded and followed by one space. Note that if it is the first entry on a line, a semicolon is sent first. When the end of the line is reached, a <CR><LF>; is issued and printing resumes. If the trace is interrupted by error or status messages, the next trace number is started on a new line. Since the test number is written prior to starting the test, the last number written will indicate the failing test.

Self-Test Summary Report

XCDST supports one level of summary. This information is printed when test execution is completed or terminated by a control character. The summary report line consists of a pass/fail indicator, followed by the XMI node number, XMI device code, and the pass count.

Two examples:

```
;     F       C       0C05       00000005
```

and

```
;     P       B       0C05       00000007
```

In the first example, the indication is that the test failed, while in the second, the test was successful. The device was node C in the first and node B in the second. In both cases, the 0C05 indicates the CIXCD, and they completed five and seven passes respectively.

Self-Test Error Reports

XCDST follows the XMI standard for RBD error reports. It supports three levels of error reporting, each one on a separate line. The three levels are:

1. XMI information (same as the summary report)
2. Error class/device type information
3. Error specific information

The level 1 error report line consists of the same four fields as the summary line: a failure indicator, the XMI node number, the CIXCD identifier, and a decimal pass count.

The level 2 error report line consists of an error class, hard error (HE) or fatal error (FE), followed by the device under test, the unit number (if applicable), and the diagnostic test number.

The level 3 report contains six fields of error specific information. The first field is the two-digit subtest number, while the remaining five fields contain eight hex digits each. The second and third fields contain the expected and actual data respectively. The fourth field (if nonzero) indicates a failing address. The fifth field is unused and will be filled with 0s. The last field is the error PC.

Example:

```
;   F          C        0C05   00000005
;  HE        XCD          XX        T01
;  07   55555555  55555554  00000800  00000000  00000000
```

This tells us that:

- For level 1: diagnostic failed, the device being tested was node C, it was a CIXCD, and it completed five passes.
- For level 2: a hard error occurred, the XCD was being tested, it failed in test 1.
- For level 3: subtest 7 failed, expected data was 55555555(hex), actual data was 55555554(hex), the failing address was 800(hex), and the error PC was 0.

5.1.2.3 Example of RBD Printout

An example of an RBD being called in and detecting an error in a VAX 9000 system follows:

```
>>>
>>>  Z 04
[  Use ^P to exit Z-MODE  ]
T/R

RBD4>
RBD4>ST 0 /TR /HE

;XCD_ST     1.00

; T01 T02 T03 T04 T05 T06 T07 T08

;  F           4       0C05   00000001
; HE         XCD        XX       T08
; 23     55555555  55545555  0000064C  00000000  00000000

RBD4>
```

5.2 Macrodiagnostics

To test the CIXCD's ability to function correctly, beyond the self-test/RBD level, there are additional macrodiagnostics available. Level 3 diagnostics include EVGEA (repair level), EVGAA and EVGAB (CI functional), and EVGAC (cluster functional). In addition, there is EVXCI, which is a level 2R diagnostic, run under the VMS operating system. Only the level 3 diagnostics are discussed here.

5.2.1 Repair-Level Diagnostic — EVGEA

EVGEA provides extensive testing of the CIXCD at the logic level and at the functional level. At the logic level, EVGEA verifies that the components are working correctly. At the functional level, EVGEA verifies that the CIXCD adapter is performing the error-free operations it is capable of performing.

The functions tested include:

- Being able to invoke XCDST and read the results
- Calculating the checksum of the EEPROM code (functional and diagnostic)
- Testing the read/write capability of the control store
- Performing a function-level test for RAM memory

Also, included in EVGEA is the EEPROM update/verification utility, which is accessed through the /SECTION qualifier of VDS.

5.2.1.1 Running EVGEA

Since EVGEA is a level 3 diagnostic, you must first load and run VDS. (Refer to the applicable system installation manual for the VDS load and run procedures.)

NOTE
If VDS is loaded through a CIXCD, VDS automatically assigns the designation PAA0 to that CIXCD. Other CIXCDs to be tested must be assigned other designations (for example, PAB0 or PAC0). Be careful to avoid the command SELECT ALL under these circumstances, as it will cause PAA0 to be tested. (Not booting through the CIXCD allows you to use the PAA0 designation and the SELECT ALL command.)

Once you have successfully loaded and run VDS, attach and select all CIXCDs to be tested. Avoid using the SELECT ALL command when VDS is booted through a CIXCD. An example of the format follows:

For VAX 6000 systems:

```
DS>  ATTACH  CIXCD  HUB  PAB0  C  3   ;C is XMI node number,
                                      ;3 is CI node number
DS>  SELECT  PAB0
DS>
```

For VAX 9000 systems:

```
DS>  ATTACH  XJA  HUB  XJA0  0  8       ;0 is XMI number,
                                        ;8 is XMI node number
DS>  ATTACH  CIXCD  XJA0  PAB0  2  3    ;2 is XMI node number,
                                        ;3 is CI node number
DS>  SELECT PAB0,XJA0
DS>
```

EVGEA is divided into 12 sections that are individually accessible through the use of the /SECTION qualifier of VDS. The sections of EVGEA and their functions are:

- DEFAULT — Tests the CIXCD with microcode loaded into the EEPROM
- MFG — Tests the CIXCD without microcode in the EEPROM
- RBD — Interfaces the user to the RBDs on the CIXCD
- UPDATE — Updates the microcode in EEPROM
- VERIFY — Verifies the microcode in the EEPROM against the load media
- RVERIFY — Verifies the primary EEPROM regions against the backup regions
- REPLACE — Replaces the backup region from the primary region
- RESTORE — Restores the primary region from the backup region
- ERRORLOG — Examines error log header information in the EEPROM
- EXAMLOG — Examines error log entry information in the EEPROM
- INIT_DCB — Initializes the error log data in the EEPROM from the keyboard
- BAR_DCB — Initializes the error log data in the EEPROM from the barcode

The diagnostics make up the DEFAULT section of EVGEA. You can run them by not using the /SECTION qualifier, or by using the qualifier and specifying the DEFAULT section. The diagnostics test the CIXCD as if the EEPROM has been initialized with valid microcode. The DEFAULT section does not destroy any microcode data contained in the EEPROM, which limits the testing of the EEPROM data and addresses.

The MFG section tests the CIXCD as if there were no valid microcode loaded, resulting in the destruction of any microcode that is loaded. The MFG section tests all addresses and data bits in the EEPROM. This section should only be run when the the UPDATE section has failed, and then you should run the INIT_DCB section to reload the module serial number and hardware revision. Run the UPDATE section again to load valid microcode into the EEPROM, and then run the diagnostics so tests that require valid microcode in EEPROM can be executed.

The RBD section provides an interface to run the RBDs, while preserving the VDS environment.

The other nine sections of EVGEA are not really diagnostics. They make up the EEPROM update/verification utility. For more information about them, refer to Section 5.2.1.5.

CAUTION
To execute the diagnostics or the DEFAULT, MFG, UPDATE, REPLACE, or RESTORE sections, the system must be in update mode:

- **For VAX 6000 systems, the lower console keyswitch must be in the update position (refer to Figure 3-6).**
- **For VAX 9000 systems, use the SET XMI_UPDATE/XMI:n ON console command. For correct command syntax, refer to the *VAX 9000 Model 200 Hardware User Guide*.**

5.2.1.2 Event Flags

EVGEA uses two of the VDS event flags. They are:

- Event flag 1 — set enables the execution of the CI external loopback subtests (test 8, subtests 10 and 11); clear inhibits execution
- Event flag 2 — set inhibits loading of failing-test information into the EEPROM; clear allows loading

5.2.1.3 EVGEA Tests

EVGEA is a repair-level diagnostic designed to test the CIXCD for functional hardware failures. It uses a bottom-up approach, with each test running only after all preceding tests have run successfully. The tests are:

Test 1: CIXCD Scan Data Path Verification

Subtest 1: Port scan data register loopback
Subtest 2: Port scan control register
Subtest 3: Port scan shift register
Subtest 4: PMCS RAM data path

Test 2: PMCS EEPROM Region Checksums

Subtest 1: Backup region functional microcode checksum
Subtest 2: Primary region functional microcode checksum
Subtest 3: Backup region self-test microcode checksum
Subtest 4: Primary region self-test microcode checksum

Test 3: PMCS RAM Memory

Test 4: PMCS EEPROM Memory[1] (MFG section only)

Test 5: CIXCD Node Reset

Subtest 1: Run self-test using XBER NRST — Check XBER status
Subtest 2: XMOV loads RAM from EEPROM — Check XMOV and RAM data

Test 6: Scan Visibility Bus CS Address Field

Test 7: XMI Device (XDEV) and Port Serial Number (PSNR) Registers

[1] Since the EEPROMs have a lifetime of approximately 10,000 write cycles, this test is not run as part of the DEFAULT program section. It is in the MFG section and must be called out specifically.

Test 8: ROM-Based Diagnostic (RBD) Interface

Subtest 1: RBD 0 — Power up self-test
Subtest 2: RBD 2 Test 1 — Port local store data integrity
Subtest 3: RBD 2 Test 2 — Port local store address independence
Subtest 4: RBD 2 Test 3 — Port local store literal addressing
Subtest 5: RBD 2 Test 4 — Port local store TCB base relative addressing
Subtest 6: RBD 3 Test 1 — Port packet buffer data integrity
Subtest 7: RBD 3 Test 2 — Port packet buffer address independence
Subtest 8: RBD 4[1] — XMI commander
Subtest 9: RBD 1 Test 1 — Verify CI jumpers
Subtest 10: RBD 1 Test 2 — External loopback on path A[2]
Subtest 11: RBD 1 Test 3 — External loopback on path B[2]

5.2.1.4 The MFG Section

The MFG section is designed to be run by manufacturing on modules that do not yet have the valid microcode loaded. It tests the EEPROM and associated logic, including the ability to write to the EEPROM. The MFG section runs tests 1, 3, and 4. This section requires that the system be in update mode (refer to the caution at the end of Section 5.2.1.1).

5.2.1.5 The EEPROM Update/Verification Utility

The EEPROM update/verification utility has nine separate sections in EVGEA: UPDATE, VERIFY, RVERIFY, REPLACE, RESTORE, INIT_DCB, BAR_DCB, ERRORLOG, and EXAMLOG. (These sections are also contained in a separate program called EVGEB, which is supplied to customers who do not have a diagnostic license.) The program sections and their functions are shown in Table 5–5.

[1] RBD 4 is designed to be run only under the control of EVGEA. Do not run RBD 4 standalone under any circumstances.

[2] Subtests 10 and 11 require CI bus loopback connectors (refer to Section 4.1.3.1) and are not run by default. To run these subtests, they must be enabled by event flag 1. Executing these subtests on a system connected to running cluster may cause unexpected errors.

Table 5–5 EEPROM Update/Verification Utility Program Sections

Section	Function
UPDATE [1]	Initializes or updates the functional and diagnostic microcode from a file on the load media.
VERIFY	Verifies the functional and diagnostic microcode against a file on the load media.
RVERIFY	Verifies the functional and diagnostic microcode primary regions against the backup regions without the use of a file on the load media.
REPLACE[1]	Replaces the backup regions from the primary regions.
RESTORE[1]	Copies the backup versions of the microcode (both diagnostic and functional) into the main areas (refer to Figure 5–1).
INIT_DCB	Initializes the diagnostic control block (DCB) and error history from the keyboard.
BAR_DCB	Initializes DCB and error history from a bar code reader.
ERRORLOG	Examines DCB and error history header.
EXAMLOG	Examines DCB and error history entries.

[1]System must be in update mode. Refer to the caution at the end of Section 5.2.1.1.

Update Functional and Diagnostic Firmware

This section provides the user with the ability to load or update the firmware stored in the PMCS EEPROM. A checksum is generated for both the functional and diagnostic microcode and stored in the DCB, then the microcode is loaded to both the primary and backup EEPROM regions. After each region is loaded, it is reread and verified to be correct.

Verify Functional and Diagnostic Firmware

This section allows the user to compare the firmware stored in the EEPROM, in the primary and backup regions, against a file on the load media, to verify its integrity.

Rverify Functional and Diagnostic Firmware

This section provides the user with the ability to verify the firmware stored in the PMCS EEPROM primary regions against the backup regions without the use of a file on the load media. The microcode in the primary and backup EEPROM regions is checked.

Replace the Backup EEPROM Regions

This section allows the user to copy the EEPROM primary regions into the backup regions. This provides the means to recover from an EEPROM data error.

Restore Functional and Diagnostic Microcode

This section allows the user to copy the EEPROM backup regions into the primary regions. This provides the means to recover from an EEPROM data error.

Initialize DCB from Keyboard

This program section is used to initialize the DCB from operator keyboard input. The information initialized follows:

- Module serial number
- Module revision
- Number of ERRORLOG entries
- Number of EEPROM write cycles

When selected for other than manufacturing use, this section gives the user the option to clear the error history buffer. The error history buffer may contain valuable information for future diagnosis of the CIXCD. Because of this, the error history buffer should only be cleared if necessary.

Initialize DCB from a Bar Code Reader

When selected, this program section is used to initialize the DCB from the bar code reader input. Refer to the previous section for details.

Examine DCB Errorlog Header Information (ERRORLOG)

This program section allows the user to view the DCB and all error history header information for this CIXCD. When executed, the following information is displayed:

- Current XMI node number
- Current date and time
- Date and time the EEPROM was last updated
- Module serial number
- Module revision
- Functional microcode revision
- Diagnostic microcode revision
- Functional microcode region checksum
- Diagnostic microcode region checksum
- Diagnostic control block checksum
- Number of ERRORLOG entries
- Number of EEPROM write cycles

Examine DCB Errorlog Entry Information (EXAMLOG)

This program section allows the user to view the DCB errorlog history information for this CIXCD.

If there are any errorlog entries, the user is then given the option of viewing either an individual entry or all errorlog entries for this CIXCD error log. (Refer to Example 5–1.)

NOTE
There is a maximum of eight entries that can be viewed. Entries 1–7 contain information about the first seven errors detected. Entry 8 contains information about the most recent error.

When executed the following informatior is displayed:

- Current XMI node number
- Number of valid errorlog entries
- Error log entry number
- Failing test/subtest number
- Failing address
- Expected data
- Actual data
- Date and time of error

NOTE
The failing test number listed in the entry refers to the diagnostic test number. Also, each section of EVGEA has an assigned number to indicate which section was under test at the time of the failure. The sections and their corresponding numbers are:

- **UPDATE: Test 12**
- **VERIFY: Test 13**
- **INIT_DCB: Test 14**
- **BAR_DCB: Test 15**
- **ERRORLOG: Test 16**
- **RESTORE: Test 17**
- **REPLACE: Test 18**
- **RVERIFY: Test 19**
- **RBD: Test [illegible]**
- **EXAMLOG: Test 25**

```
DS> START/SECTION=EXAM

.. Program: EVGEA CIXCD Functional Level 3 Diag , revision 2.0, 25 tests,
   at 14:23:26.74.
Testing: _PAB0

CIXCD EEPROM examine ERRORLOG entry utility

The current CIXCD XMI node number is 04

Number of valid ERRORLOG entries = 05

Examine error entry # (RETURN = All) [(00000000), 00000001-00000008(X)] 5

Error Log entry 05:
        Failing Test Number- - - 8, Subtest  8
        Failing Address- - - - - 00000005
        Expected Data- - - - - - 00000000  00000000  00000034
        Actual Data- - - - - - - 00000000  00000000  00000000
        Date and Time of Error - 28-MAR-1990 17:56:20.22

.. End of run, 0 errors detected, pass count is 1,
   time is 29-MAR-1990 07:43:06.12
DS>
```

Example 5-1 Sample Printout of Exam Section

5.2.1.6 EEPROM Memory Map

Figure 5-1 shows the layout of the EEPROM.

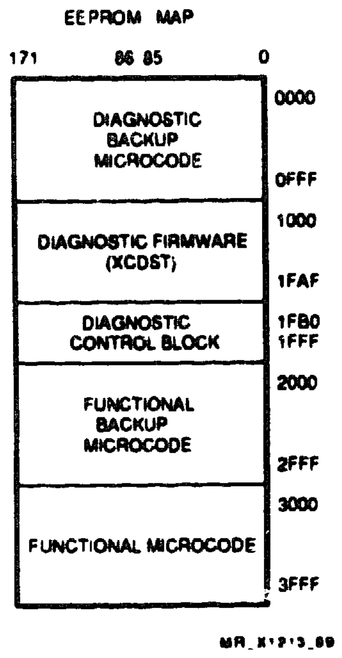


Figure 5–1 EEPROM Memory Map

5.2.1.7 EVGEA Error Messages

When EVGEA detects an error, it prints an error message. Example 5–2 shows the format.

```
******  CIXCD Functional Diag  --  ZZ-EVGEA  --  1.0  ******

Pass 1,  Test 1,  Subtest 3,  error 1,  18-MAY-1990 14:36:47.83
Hard error while testing PAB0:  CIXCD Port Scan Shift Register Error

 Address            Expected           Received            XOR
00000000 (X)       FFFFFC00 (X)       FFFFFB00 (X)       00000100 (X)

******   End of Hard Error number 1   ******
```

Example 5–2 EVGEA Error Message

5.2.2 CI Functional Diagnostics — EVGAA, EVGAB

These two diagnostics are the standard CI bus adapter diagnostics, upgraded to include the CIXCD.

The tests that make up these diagnostics are:

EVGAA:

Test 1: Cluster Configuration
Test 2: SETCKT Test with Various Masks and M_values
Test 3: SETCKT Test for Each Valid Port
Test 4: SETCKT Test for Invalid Port
Test 5: REQID Test
Test 6: REQID Test with 6 Packets of DGFQ
Test 7: Datagram Discard Test
Test 8: RESPONSE Queue Available Interrupt Test
Test 9: Send Datagram -SNDDG- Test
Test 10: SNDMSG Test with No Virtual Circuit Set
Test 11: Send Message Test, Crossing Page Boundary
Test 12: Message Length Test
Test 13: Packet Size Violation Test
Test 14: Send Loopback -SNDLB- Test
Test 15: SNDLB Test, Full Buffer Path A
Test 16: SNDLB Test, Full Buffer Path B
Test 17: SNDLB Test, Both Paths

EVGAB:

Test 1: Send Data Test, with Offset Combinations
Test 2: Request Data Test, with Offset Combinations
Test 3: Invalidate Translation Cache Test
Test 4: SNDMDAT Test, Enabled/Maintenance State
Test 5: SNDMDAT Test, Enabled State
Test 6: REQMDAT Test, Enabled/Maintenance State
Test 7: REQMDAT Test, Enabled State
Test 8: Send RESET Test, Enabled State
Test 9: Queue Contention Test
Test 10: Buffer Read Access Test
Test 11: Buffer Write Access Test
Test 12: Write to Global Buffer Test

5.2.3 Cluster Functional Diagnostic — EVGAC

EVGAC is a functional port-to-port exerciser, similar to EVXCI, but while EVXCI is a level-2 exerciser run under the VMS operating system, EVGAC is a level-3 standalone diagnostic run under VDS. The faults it detects are of a communication and data corruption type. It must be run under VDS on an inactive CI cluster, and the adapter microcode must be on the same medium as the diagnostic. Also, it assumes that both EVGAA and EVGAB have run successfully.

5.2.3.1 Event Flags

EVGAC uses the VDS event flags to determine how the tests should be run. There are four event flags.

- Event flag 1: Set causes the loading of microcode; clear prevents the loading. When set during CIXCD testing, it causes the CIXCD to set the NRST bit (bit <30>) of XBER, initiating a complete power-up reset.
- Event flag 2: Set displays the following (while clear does not display):
  - CI configuration (test 1)
  - Total number of usable pages in memory
  - Changes in virtual circuit state
  - Port to which traffic is being sent (tests 3:8)

- Event flag 3: Set for displaying confirmation received (CNFREC) and data received (DATREC) packets; clear does not display.
- Event flag 4: Set for displaying packets taken from the response queue which contain a nonzero error status; clear does not display.

Note that the RUN command clears all event flags, as does the LOAD command. To use them, you must first load EVGAC, set the desired event flags, and then start the diagnostic.

5.2.3.2 Event Tracing

In EVGAC, the user is allowed to trace specific events and to enable/disable specific program routines. This is done through the use of the 32-bit program parameter register (only the lower 12 bits are used). Refer to Figure 5-2. EVGAC prompts the user for input into this register, with the default being all bits clear. If the operator flag is cleared, the diagnostic does not prompt for input and uses the register with all bits clear. If trace bits are set by the user (Table 5-6), the interrupt-driven print routines might interfere with other common print routines.

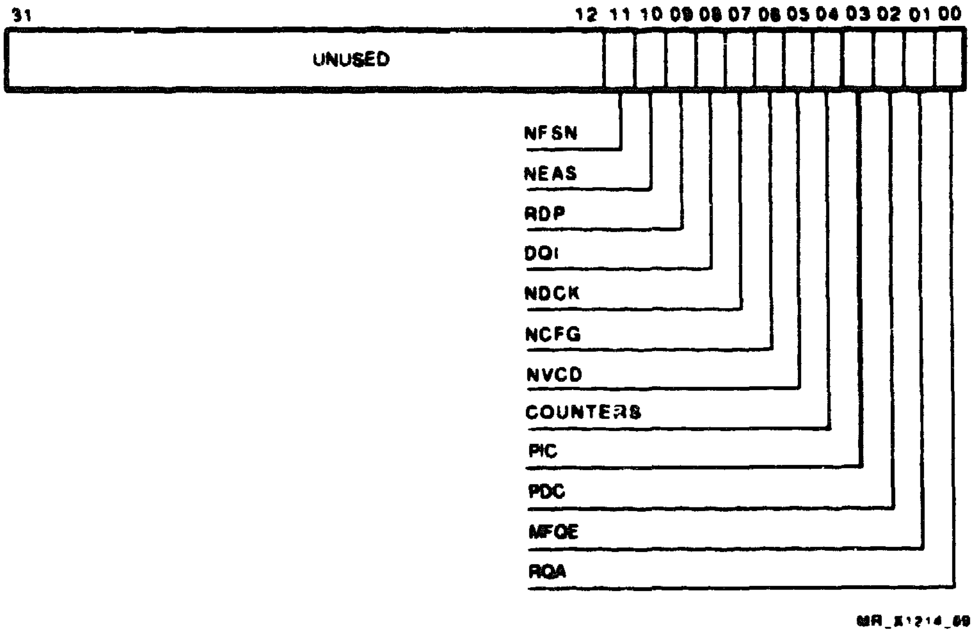


Figure 5-2 Program Parameter Register

Table 5–6 Trace Bit Field Definitions

Bit	Name	Function
<31:12>	Unused	Unused.
<11>	NFSN	If set, this causes the FSN bit in the local adapter's virtual circuit descriptor entry in the VCDT to clear.
<10>	NEAS	If set, this causes the EAS bit in the local adapter's virtual circuit descriptor entry in the VCDT to clear.
<09>	RDP	If set, this causes the RDP bit in the local adapter's virtual circuit descriptor entry in the VCDT to set.
<08>	DQI	If set, this causes the DQI bit in the local adapter's virtual circuit descriptor entry in the VCDT to set.
<07>	NDCK	If set, this disables the data-checking routines.
<06>	NCFG	If set, this disables the running of the configuration routine at the beginning of each test. The exception to this is that the configuration routine will always be run in test 1.
<05>	NVCD	If clear, this allows the program to re-establish virtual circuits between the local and remote nodes when a virtual circuit is dropped. If set, will inhibit program re-establishment of virtual circuits for that test pass.
<04>	COUNTERS	(Specific to test 8) If set, this causes the program to read and display the counters of the local adapter.
<03>	PIC	If set, this causes a message to be displayed when a port initialization complete interrupt occurs.
<02>	PDC	If set, this causes a message to be displayed when a port disable complete interrupt occurs.
<01>	MFQE	If set, this causes a message to be displayed when a message free queue empty interrupt occurs.

Table 5–6 (Cont.) Trace Bit Field Definitions

Bit	Name	Function
<00>	RQA	If set, this causes a message to be displayed when a response queue available interrupt occurs.

5.2.3.3 Program Parameters

The user has the ability to control the diagnostic by setting up the program's parameters. These parameters can be defined in one of three ways:

- Program default values
- User entering the values of the console device at program prompts
- Parameter file

The function of each parameter and the default values are explained in Table 5–7.

Table 5–7 EVGAC Program Parameters

Parameter	Default[1]	Function
minport	CIXCD's CI port number	The minimum port number to which the diagnostic sends test packets. The limit is the maximum cluster size found in the port parameter register (PPR). Range: 0 to PPR
maxport	CIXCD's CI port number	The maximum port number to which the diagnostic sends test packets. The limit is the maximum cluster size found in the PPR. Range: minport to PPR
sanity	0	Sanity timer value, with limits of 0 to 99.
maxcmd	47	The number of commands the program sends to each node per pass of the activity test (test 8) ranging from 0 through 400.

[1]All default values are in decimal.

Table 5–7 (Cont.) EVGAC Program Parameters

Parameter	Default[1]	Function
dgfq	50	The number of datagram free queue entries, with limits from 0 to 2048.[2]
msgfq	50	The initial number of message free queue entries, with limits from 0 to 2048. This can be considered dynamic; that is, when the program receives an MFQE interrupt from the CIXCD, it tries to allocate five queue entries to place onto the message free queue. If EVGAC is unsuccessful in allocating the buffers, it aborts.
entrysize	Internal buffer length	The maximum datagram and message queue size, which is used by EVGAC if it is less than the internal buffer length found in the PPR. If it is greater than the internal buffer length, it defaults to the value in the PPR.
nbuffmin	512	Minimum size of named buffers, with limits from 1 to 2147480000. This value can be dynamically changed, if insufficient host memory is available.[3]
nbuffmax	13739	Maximum size of named buffers, with limits from *nbuffmin* to 2147483647. This value can be dynamically changed, if insufficient host memory is available.[3]
pm	PPR value	Packet multiple. This value is used if it is less than or equal to the value calculated from the PPR. The value in the PPR is used if it is greater.

[1]All default values are in decimal.

[2]If a less than acceptable amount of datagram free queues are created, it will, in effect, inhibit the port from receiving necessary packets from remote ports. Try increasing the number if tests are failing due to unreceipt of datagram type packets.

[3]Any dynamic changes are displayed on the console.

5.2.3.4 Support Files

There are two support files that the user can set up to pass parameters to EVGAC: a parameter file (PARAMETER.PAR) and a pattern file (PATTERN.PTN). At runtime, the diagnostic allows the user to select whether to use either, both, or neither of these files. The diagnostic also offers the user the opportunity at this time to pass parameters directly from the console.

If the VDS operator flag is clear, the parameter and pattern files are not used, no prompt is issued, and default values are used.

PARAMETER.PAR, the Parameter File

PARAMETER.PAR, created by the user and copied to the load media, allows the user to set program parameters by reading in a file, rather than having to input each value using the console device or using the program default values. It must be done exactly as described below.

The parameter file is an ASCII file of eight characters per line with each line representing one parameter. Each line is read in by the program, and that hexadecimal value (remember that the defaults were listed in decimal) is placed in the appropriate parameter location. The parameters need to be placed in exact order as described in Table 5–8. Program default values are used if the values entered exceed the maximum values of the adapter specified in Table 5–7.

Table 5–8 Parameter File Structure

Line	Parameter
1	minport
2	maxport
3	sanity
4	maxcmd
5	dgfq
6	msgfq
7	entrysize
8	nbuffmin
9	nbuffmax
10	pm

If the parameter file cannot be accessed or the format of the file is incorrect, an error message is generated and the user is then prompted to either input the parameters or use default values. If the parameter file contains an invalid parameter, the user is prompted to either use default values or abort the program. Example 5–3 shows a sample parameter file.

```
00000000
00000010
00000000
0000005D
00000064
00000064
000003F8
00000200
00007C1B
00000000
```

Example 5–3 Sample Parameter File

PATTERN.PTN, the Pattern File

The pattern file (Example 5–4) allows the user to select the text to be used in all message, datagram, and named buffer transfers. It must be created by the user, in the format described below, and copied to the load media.

The pattern file is an ASCII file of eight characters per line, where each line is read and stored in a data area. The pattern file can be any size greater than one 8-character line, up to 1024 bytes. If the program detects an end-of-file condition before the data area is complete, it closes the pattern file and fills the remainder of the data area with characters previously read. The size of the pattern data area to be filled is 1024 bytes.

If the pattern file cannot be accessed or the format of the file is incorrect, a message is generated and the program uses a default pattern.

```
!1!1!1!1
2@2@2@2@
#3#3#3#3
4$4$4$4$
%5%5 5 5
6^6^6^6^
&7&7&7&7
8*8*8*8*
(9(9(9(9
0)0)0)0)
_-_-_-_-
=+=+=+=+
QqQqQqQq
wWwWwWwW
EeEeEeEe
rRrRrRrR
TtTtTtTt
yYyYyYyY
UuUuUuUu
iIiIiIiI
OoOoOoOo
pPpPpPpP
{[{[{[{[
]}]}]}]}
AaAaAaAa
sSsSsSsS
DdDdDdDd
fFfFfFfF
GgGgGgGg
hHhHhHhH
JjJjJjJj
kKkKkKkK
LlLlLlLl
;:;:;:;:
"'"'"'"'
\|\|\|\|
><><><><
zZzZzZzZ
XxXxXxXx
cCcCcCcC
VvVvVvVv
bBbBbBbB
NnNnNnNn
mMmMmMmM
```

Example 5–4 Sample Pattern File

5.2.3.5 Program Tests

At the start of each test, a cluster configuration routine is displayed. The configuration routine first initializes the configuration table memory area. Then it sends a REQID datagram on *both* paths to each node on the CI bus, limited between the minimum and maximum port number parameters passed to the program. The program waits until it receives responses, error or nonerror, for the REQID packet it sent. On receipt of an IDREC packet, the configuration table is searched for an entry of the port identified in the packet. If no entry is found, an entry is created to include the data found in the IDREC packet. If the entry is found, the entry is updated using the contents of the IDREC packet.

After sending and receiving the necessary REQID and IDREC packets, the cluster configuration routine sends a parameter request packet to each remote port requesting their datagram, message, DMA buffer sizes, and their virtual circuit descriptor (VCD) entry. The message, datagram, and DMA sizes are placed in the configuration entry for the remote port.

It is possible that a datagram packet, such as these, could be lost. A count and wait loop is exercised as to not be caught in a timeless loop and to ensure that all remote ports have sufficient time to respond to the local port's requests.

The above routine is always executed in test 1, but not necessarily in other tests. This is dependent on the CONFIG bit in the program parameter register. If set, the program allows the above routine to execute only once for each pass of the program. The configuration table is displayed to the console if event flag 2 is set.

Before each test, excluding test 1 and test 2, the program checks the configuration entry for the node currently being tested. If the port did not return a parameter returned packet, the port is not tested. When a path is marked bad, no packets are sent to the port along that path. Message, datagram, and DMA sizes are compared against the local port's values to restrict the packet and buffer sizes to the lesser of the two values between the nodes.

The tests are:

Test 1: Local Configuration Test
Test 2: Local Adapter Test

Subtest 1: Local adapter loopback
Subtest 2: Local adapter datagram
Subtest 3: Local adapter message

Test 3: Datagram Test
Test 4: Virtual Circuits Test

Test 5: Message Test
Test 6: Multiple Message Test
Test 7: Write and Read Buffer Test
Test 8: Activity Test

Example 5–5 shows a full listing of EVGAC tests and responses to these tests.

```
DS> st/pass:1

.. Program: EVGAC - CI Functional Exerciser, revision 1.0, 8 tests
   at 11:42:21.26.
Testing _PABC

Event Flag 1 Microcode Loading
Event Flag 2 Miscellaneous Status Messages
Event Flag 3 Datrec and Cnfrec
Event Flag 4 Display Bad Status Packets

-*- Nfsn Neas Rdp Dqi -*- Ndck Ncfg Nvcd Cntr -*- PIC PDC MFQE RQA -*-

Program Parameter Register. > [(00000000), 00000000-00000FFF (X)]
Use the Pattern File? > [(No), Yes]
Use the Parameter File? > [(No), Yes]
Modify Parameters? > [(No), Yes]

Test 1:  Local Configuration
Test 2:  Local Adapter Test
Test 3:  Datagram Test
Test 4:  Virtual Circuits Test
Test 5:  Message Test
Test 6:  Multiple Message Test
Test 7:  Write/Read Buffer Test
Test 8:  Activity Test
********  EVGAC - CI FUNCTIONAL EXERCISER - 1.1  ********
Pass 1, test 8, subtest 0, error 6, 31-AUG-1989 11:46:59.31
Soft error while testing PABO: Buffered Data Error.

Port Number:      00000006
Offset:           00000E00
Expected:         305A3159
Received.         AAAAAAAA

********  End of Soft error number 6  ********

.. Halt on error at PC 0000A36B (X)
DS> cont
..Continuing from 0000A36B

.. End of run,  0 errors detected,  pass count is 1,
   time is 31-MAR-1990 11:47:17.98
DS>
```

Example 5–5 EVGAC Full Listing

CHAPTER 6

6 Functional Description

This chapter provides a description of how the CIXCD port adapter works. It first gives an overview of the functions, and then describes each function in more detail.

The sections include:

- Overview
- XMI corner
- XMI interface logic and data movers
- Port microprocessor
- CI control logic and packet memory
- CI corner

6.1 Overview

The CIXCD port is an intelligent controller that connects the high-speed CI bus to the XMI bus. It implements the VAX-11 CI port architecture with its own integral microprocessor and EEPROM/RAM control store. The CIXCD processes commands found on the port queue block (PQB) and packets received from the CI bus. The CIXCD supports dual CI paths in resequencing dual path (RDP) operation. Refer to Figure 6–1 for a simplified CIXCD block diagram.

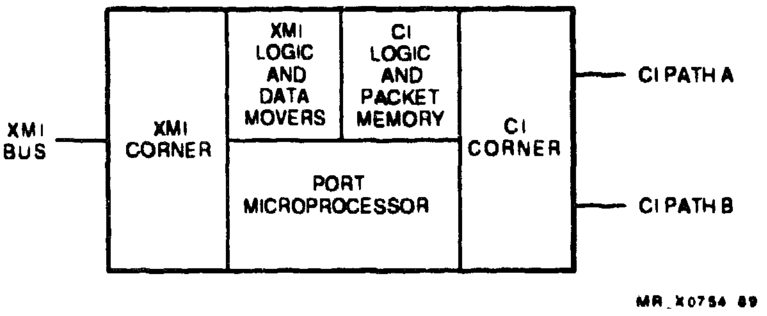


Figure 6–1 CIXCD Simplified Block Diagram

The CIXCD can be logically divided into five parts (Figure 6–2):

- **XMI corner** — This logical partition contains the XMI specific XCLOCK and XLATCH chips that interface the CIXCD to the XMI backplane bus. It consists of seven XLATCH chips and one XCLOCK chip.
- **XMI logic and data movers** — This partition consists of the XMOV gate array. The logic contained in this XMOV gate array controls and responds to the XMI corner. The XMI data path contains 64 data and 2 parity bits, organized into two 32-bit registers when dealing with the port microprocessor. The interrupt logic, also in this section, supports one interrupt vector. Two 32-bit-wide data movers are capable of 40-Mbytes bandwidth combined. Once started by the port microprocessor, the movers are free running. Each data mover transfers in a single direction. Mover A does XMI read transactions and mover B does XMI write transactions.

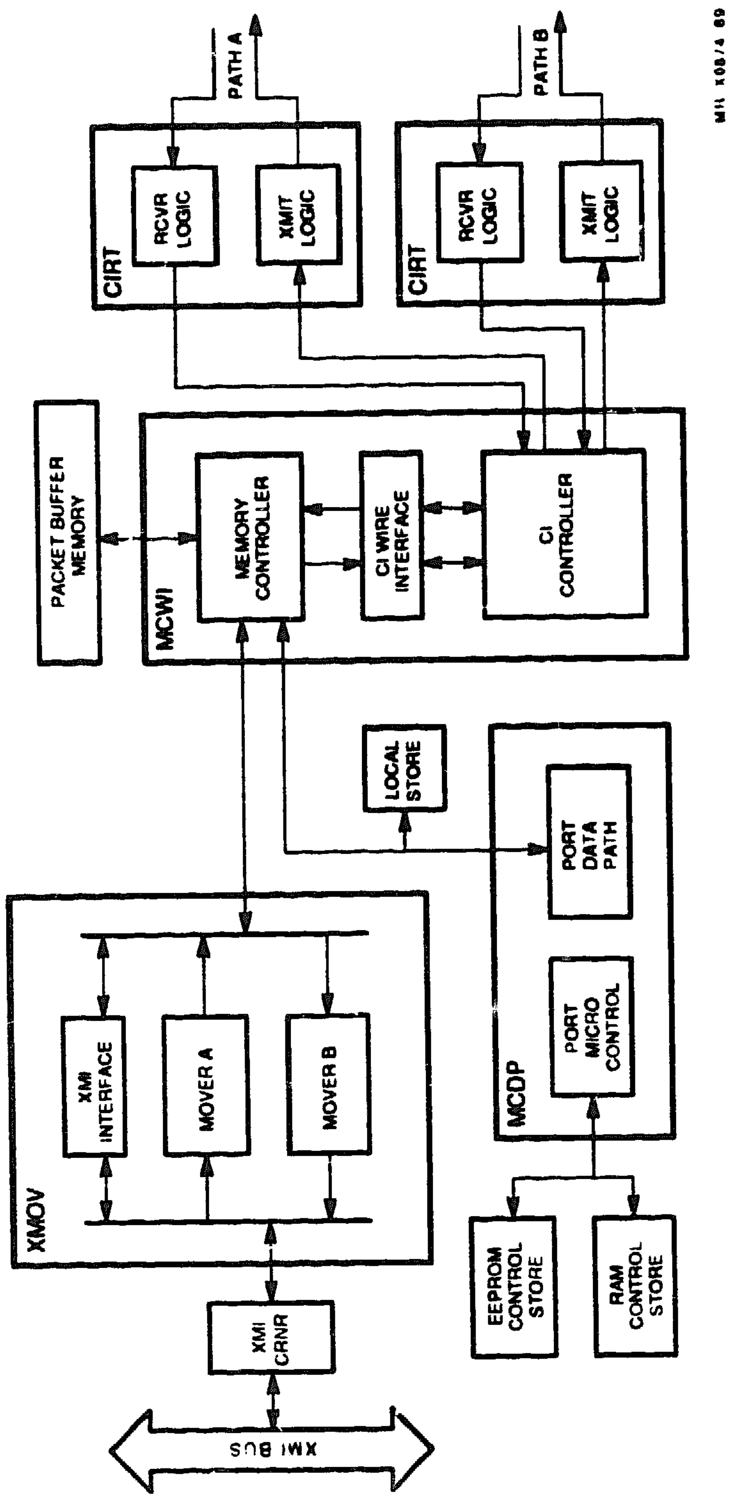


Figure 6–2 CIXCD Block Diagram

- **Port microprocessor** — A custom designed microprocessor and microsequencer allow the execution of ALU operations every 64 ns and next address calculations every 128 ns. Addressing for the microcode is 8K × 86. The port microprocessor implements a 32-bit-wide data path with internal parity and contains 32 GPRs and a 16-location microaddress stack, and has 8K × 33 of local storage available for use. This logical partition consists of the MCDP gate array, control store random access memory (CSRAM), control store electrically erasable and programmable read only memory (EEPROM), and the local store random access memory (LSRAM).
- **CI control logic and packet memory** — This partition consists of the MCWI gate array and its associated PB RAMs. The MCWI contains the logic that implements the CI protocol and controls the CI interface. PB RAM access requests from the CI wire, data movers, and port microprocessor are arbitrated and controlled by the buffer access control contained in the MCWI gate array.
- **CI corner** — This partition contains two independent interfaces to the CI wires. This logic performs Manchester encoding, clock from data separation, and byte framing and synchronization. This logical partition consists of two CIRT gate arrays, the header card, a 140-MHz oscillator, a hybrid receiver, a set of CI wire drivers, and four transformers.

6.1.1 XMI Responder

The CIXCD responds to the following XMI transactions:

- LW read (READ)
- LW interlock read (IREAD)
- LW unlock write mask (UWMASK)
- LW write mask (WMASK)
- Interrupt acknowledge (IDENT)

6.1.2 XMI Commander

The CIXCD is capable of generating the following XMI transactions:

- LW/QW/OW READ/IREAD
- HW READ/READ MORE
- LW/QW/OW WMASK/UWMASK
- HW WMASK/WRITE MORE
- Interrupt (INTR)
- IDENT (for diagnostic purposes)
- Implied vector interrupt (IVINTR) (host register writes to invalid address)

6.2 XMI Corner

The XMI corner is a 1.8 in × 5 in area on both sides of the module that provides all the components necessary to connect to the XMI bus. All components in the XMI corner are surface-mounted. Seven XLATCH chips in the corner provide the data and control signal interface to the XMI bus, while a single XCLOCK chip provides the six-phase clocking system used by the CIXCD module.

6.2.1 XCLOCK

The XMI clock decoder chip, called the XCLOCK, is a CMOS integrated circuit that provides all required clock decoding for XMI nodes. It also drives and receives the XMI CNF lines and XMI DC LO L line. The following list summarizes the highlights of the XCLOCK chip:

- Provides low-skew clock decoding of XMI backplane clocks.
- Single XCLOCK provides a complete family of clocks synchronous with the XMI bus. The XLATCH clocks and output enables are driven by a set of XCLOCK lines. This provides for consistent loading and consistent skew for bus timing calculation purposes. Another set of clocks are provided for use by the CIXCD sea of gates (SOG) arrays.
- Provides CMOS clock levels that permit the use of high-performance clock input buffers on the SOG arrays.

6.2.2 XLATCH

The XMI interface chip, called the XLATCH, is a CMOS integrated circuit that provides the primary interface to the XMI bus. The following list summarizes the highlights of the XLATCH chip:

- Provides a high-performance interface to the XMI bus.
- Seven chips provide a complete XMI interface.
- Uses "reduced-swing" CMOS levels (voltage divider techniques) to reduce power consumption and peak currents, and improve propagation delay times.
- Provides CMOS levels on the CIXCD side, permitting the use of high-performance CMOS input buffers in the interface gate arrays.

6.3 XMI Interface Logic and Data Movers

The XMOV array consists of the XMI interface logic and dual data movers. The interface contains the required XMI registers, XMI arbitration, protocol checking (XMI and port errors), and commander/responder sequencing. The data movers are two register files with state machine control that allows high-speed transfers of packets between the PB memory and the system memory. On memory reads, the 64-bit XMI data word is unpacked to two 32-bit words for the PB controller. On memory writes, a 64-bit XMI word is assembled from the two 32-bit PB memory words.

6.3.1 XMI Interface Logic

The XMI interface performs the following functions:

- Monitors the XMI for transactions addressed to the port. Their occurrence is independent of activity in the port and is initiated by a host. The XMI interface will respond to the entire 512 Kbyte address block (node space) for the port. These addresses will be decoded and can point to registers maintained locally in the XMOV array (such as, XMI required registers) or registers maintained by the port processor (this includes local store access). In the latter case, the XMI interface notifies the port processor that a register read or write needs to be completed. The XMI interface treats all node space accesses as longword operations; interlocked read accesses will be treated the same as ordinary reads.

- Initiates an XMI transaction as requested by either the port processor or one of the two data movers. The port processor can initiate READ, WMASK, IREAD, and UWMASK commands in lengths up to octawords, as well as INTR cycles. Mover B can initiate octaword WMASK, hexword WMASK (if enabled), and mover A can initiate hexword READ commands.
- Generates interrupts as a result of detected errors or port processor requests and responds to IDENT with the appropriate vector information.

6.3.2 Data Movers

The CIXCD contains two data movers. One data mover reads host memory and writes the transmit PB (mover A). The other data mover goes in the opposite direction, emptying the receive packet buffer into host memory (mover B).

The operation of the movers is under port microcontrol. When the port processor has determined that a packet is ready to be transferred from or to XMI memory, it initializes the appropriate mover with the packet's starting XMI physical memory address and the number of bytes to move. The mover is also loaded with the XMI address of the next page, so when a packet crosses a page boundary it may continue transferring to or from that address. Once started, the mover continues transferring a packet until completion, interrupting the port processor only to notify it that the next page register has been emptied, or that the transfer completed or aborted due to a fatal error. Each mover, as well as the port processor commander, uses separate XMI ID codes, thus enabling each to operate independently.

6.4 Port Microprocessor

The microcontrol and data path (MCDP) gate array contains a 32-bit microprocessor, custom designed for the CIXCD. The main parts of the microprocessor are:

- Processor data path, which is centered on a 32-bit-wide arithmetic logic unit (ALU)
- Port microcontrol (PMC) microsequencer, a custom designed microsequencer with a 4K × 172-bit microaddress space.

6.4.1 Processor Data Path

The processor data path is designed to process data as necessary, with the main functionality provided by the 32-bit-wide ALU. There are two parallel paths in the ALU: one providing arithmetic functions and the other performing either Boolean operations or barrel shifts. The correct path for the required operation is selected just prior to ALU output. The output of the ALU is clocked into the result register and then is sent directly to the PORT IB, or it can be channeled back into the ALU using various paths for further manipulation.

The data feeding into the ALU comes from one of three sources: its own output (from the result register), the PORT IB (either directly or buffered), or the microword literal field (also either directly or buffered). Part of the buffering available to the processor data path is the set of multiported GPRs (32 locations × 33 bits).

6.4.2 Port Microcontrol

The PMC microsequencer uses a control store of 4K locations by 172 bits, producing a new microaddress and its resulting microword every 128 ns (two XCI cycles). However, the microwords are organized so there are two 86-bit fields (first field and second field), which are read out of the control store in sequence every 64 ns. Since the bit definitions of both fields are identical (except for next address information), the MCDP can perform an ALU operation every 64 ns, while calculating next addresses at a rate of 128 ns.

The first field of the microword contains a NEXT_ADDRESS_FIELD (NAF) combined with branch/dispatch conditions to determine the next microaddress. In addition to normal sequencing, branches and interrupts are allowed, giving the PMC the ability to handle events asynchronously. There is a microaddress stack that is used to allow the normal thread to be picked up after the interrupts have been handled.

6.4.3 Control Store RAMs/EEPROMs

The control store, which is external to the MCDP, consists of both EEPROMs (used for on-board storage of the microcode) and RAMs (which, at 25 ns, provide fast execution). At power-up, the EEPROM data is copied into the faster RAMs using the MCDP's copy mode.

The EEPROMs can have in-system field microcode updates using the special MCDP scan path. The upgrading of the EEPROMs is done under diagnostic control.

The CS RAM microword fields are fed to the PMC across the PMCS natural bus, which is an 86-bit-wide bidirectional bus. The natural bus is also used during copy mode to load the RAMs with the EEPROM data and under diagnostic control when the RAMs can be loaded or the EEPROMs updated.

6.4.4 Local Store

The local store RAMs are external to the MCDP gate array and are used for a virtual circuit descriptor table (VCDT), which is required by the CI bus. The local store is an 8K × 33-bit RAM and is also used for software-implemented registers. The processor accesses the local store data across the PORT IB and addresses it across the local store address lines.

The port driver has access to local store when the port is in the unitialized state. Local store is mapped into the port's XMI node space addresses. An XMI read or write request is all that is required to access local store. Local store address 0 corresponds to XMI node space address 2000, 1 corresponds to 2004, up to 1FFF corresponds to FFFC.

6.5 CI Control Logic and Packet Memory

The memory controller wire interface (MCWI) gate array controls all of the functions associated with the CI corner. This includes the CI interface protocol, CI arbitration, and reporting of CI corner transaction status to the port processor. It also controls the packet buffer address and data lines and forms the data path between the PB and the CI bus. There are three functional blocks that make up the MCWI: the memory controller (CMEM), the CI wire interface (CWIN), and the CI controller (CIC).

6.5.1 Packet Buffer Memory Organization

The MCWI has access to and control over the external 8K × 33-bit RAM that is used as a PB. It does this by providing both read and output enable controls to the PB to establish the direction of the transfer, as well as the address and data lines for reading and writing the memory.

The MCWI maintains independent packet buffer address registers for each device or piece of logic that requires access to the PB. They are for the port processor, mover A, mover B, zone 0, zone 1, Xmit path A, and Xmit path B.

Address registers are automatically incremented after a PB access is made by the associated function. This allows multiple, simultaneous block transfers to and from the PB, with minimum processor overhead. All address registers consist of an 11-bit loadable counter (bits 0 through 10), and two simple register bits (bits 11 and 12). This results in a wrap boundary of 2K longwords, with bits 11 and 12 defining which quadrant of the memory is being addressed. Refer to Figure 6–3.

In this way, the starting address of any PB transaction defines which quadrant is being accessed, while subsequent accesses and resulting automatic increments follow a 2K ring buffer format.

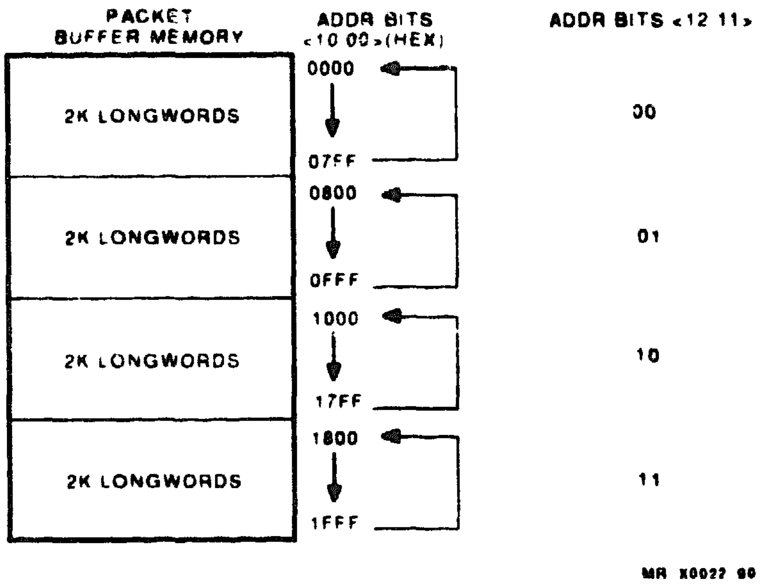


Figure 6–3 Packet Buffer Format

6.5.2 Memory Controller

The packet buffer memory is used as a temporary store for packets of data that have just been received from the CI bus or are about to be transmitted over the CI bus. Consequently, there is a requirement for the data movers, the CI interface, and the port processor to have access to this memory at various times. The function of the memory controller is to supply the packet buffer memory with address and data direction information, and to establish a data path between the packet buffer memory and the appropriate functional block within the CIXCD. Data is transferred to and from the PB over the PB data bus with longword parity while the PB is addressed by the PB address lines.

6.5.3 CI Wire Interface

The purpose of this section of logic is to interface the CI controller to the packet buffer memory through CMEM. This involves the reformatting of data for transmission, from 32-bit longwords that are stored in the packet buffer memory to 8-bit bytes that are handled by the CI controller; and the reverse for receptions. Also during this process, the data must be transferred between two asynchronous clocks: the port side uses the XMI based 64-ns clock, while the CI controller side uses the 114-ns Xmit clock (XCLK). Sufficient buffering is built in to ensure an uninterrupted flow of data on or off the CI bus when allowing for delays caused by access availability of the PB and clock synchronization.

CWIN also contains the logic that controls the organization of the packet buffer memory. The 32 Kbyte of packet buffer RAMs are divided into 16 Kbyte for transmit and 16 Kbyte for receive, with no fixed size for a single entry.

6.5.4 CI Controller

The port processor controls the CI interface using register reads and writes. These transactions are conveyed to the MCWI across the PORT IB, and addressed by the literal address lines, with the direction indicated by the level of the signal MCDP_PROC_WRITE_H.

The CI controller formats and controls the data that is passed during a transmit or receive transaction. In addition, the CIC stores the status information for these transactions.

The CIC contains a transmitter block, receiver block, and control block. The transmitter formats and checks the data packet as it is transmitted to the CI bus. The receiver checks and stores data as it is received from the CI bus. Both the transmitter and receiver function under the direction of the control block. The control is split up into three separate control units. The transmitter control unit is responsible for the control of the transmitter block as well as other aspects of a transmission, which include CI arbitration, timeouts, and status. The receiver control unit is responsible for the control of the receiver block. The command control unit combines with CWIN to interface to the port processor, receiving command information that is integrated into the control block.

Status information for transmit and receive transactions are stored in the status register.

There are two separate CIC functions implemented in the MCWI, one for each CI path.

6.6 CI Corner

The CI corner of the CIXCD consists of two CIRT gate arrays, one header card containing a hybrid receiver chip, a pair of 10192 drivers, four transformers (two of which are on the header card), and a loopback multiplexer.

This portion of the CIXCD provides the actual interface to the CI bus and is capable of servicing a dual-path CI system. This allows for the simultaneous use of both paths to transmit or receive independently of each other.

Two of the primary functions carried out by the CI corner are:

1. Transmit data from the MCWI gate array over the CI bus. The data is converted to ECL logic levels, serialized, Manchester encoded, and sent out over the CI bus.

2. Receive data from the CI bus and pass it to the MCWI gate array. The CI interface detects the presence of traffic on the CI bus. If the CI interface is able to receive data, the MCWI gate array will enable the appropriate CI receive/transmit (CIRT) gate array to decode and deserialize the incoming data. The data is passed from the CIRT gate array to the MCWI gate array.

6.6.1 Receiver Hybrid

On the header card, the low-level, high-speed CI signals are detected and received through a thick-film, chip-and-wire hybrid, containing four analog comparators that detect the CI signal and amplify it to nominal ECL levels. The carrier detect circuit monitors the signal level on the CI wire and asserts the appropriate carrier detect output when the CI signal amplitude exceeds a predetermined level. The receive circuit amplifies the CI signal and converts it to standard ECL levels.

6.6.2 CI Receive/Transmit Gate Array

The CIRT gate array provides the CIXCD with the capability of servicing a CI path, with one transmit interface and one receive interface. The receive path is Manchester-encoded serial data, which is deserialized into byte-wide data. Odd parity is generated on the byte-wide receive data. The transmit path is nibble-wide data, which is serialized into Manchester-encoded serial data. Four data bits are transmitted to the CI every 57 ns, and eight data bits are received every 114 ns.

The CIRT gate array synchronizes and locks the incoming receive data to the local 114-ns master-byte clock. The incoming Manchester-formatted CI receive data is decoded and deserialized in the CIRT gate array. The array supplies byte-wide parallel data with parity to the port interface. On a transmit, the data is serialized, Manchester-encoded, and sent out over the CI bus. The serializer inputs nibble-wide transmit data and shifts out serial data at 70 Mbits/s. The serial data is Manchester-encoded and driven out on either the CI path or the maintenance loop path as directed by the port interface.

An internal maintenance loop mode is provided to allow a node to transmit a packet to itself. In the internal maintenance loop operation, the data from the transmit channel is looped back into the receive channel. Internal loopback forces the CIRT gate array to receive its own transmissions.

APPENDIX A

A
CIXCD Registers

This appendix describes the CIXCD registers accessible from the console.

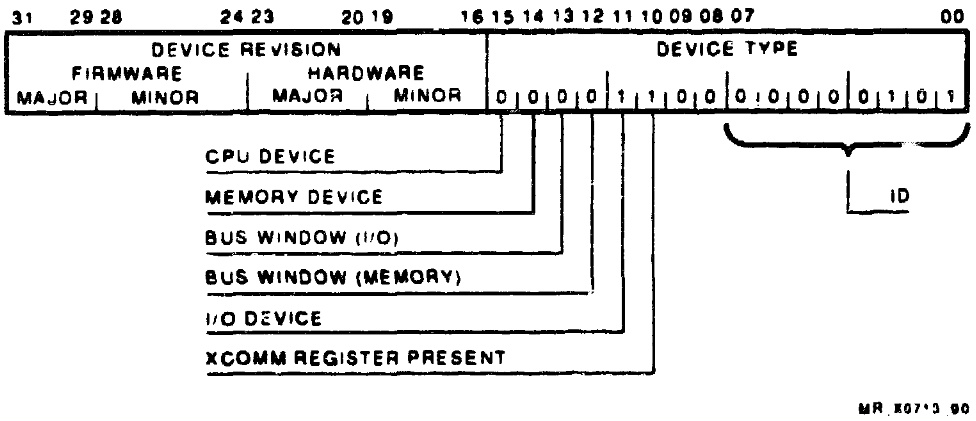


Figure A–1 XMI Device Register (XDEV) Offset = 00000

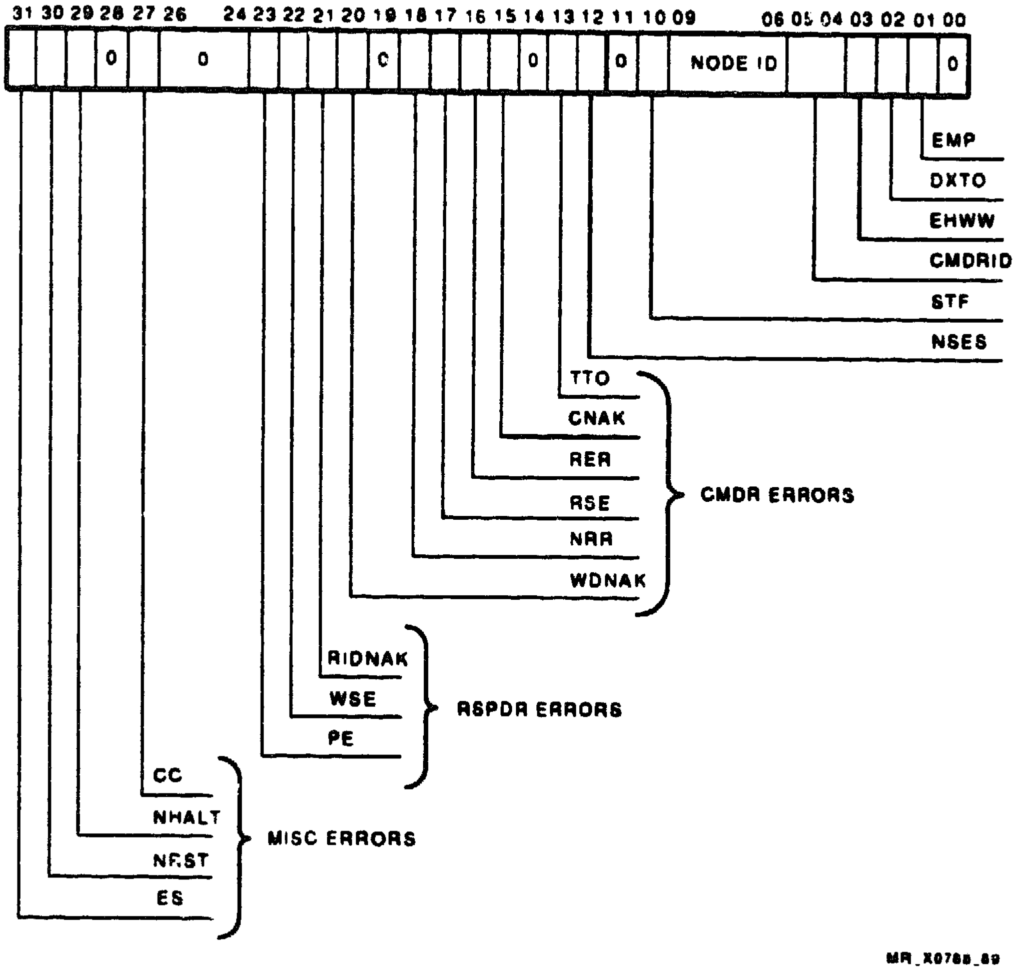


Figure A-2 XMI Bus Error Register (XBER) Offset = 00004

Bit	Name	Description
<31>	ES	Error summary (RO:RO) — This bit represents the logical-OR of the error bits in this register. These bits are: CC, PE, WSE, RIDNAK, WDNAK, NRR, RSE, RER, CNAK, TTO, and NSES. When this bit is set, an XMI interrupt will be generated, using IVIR an INTDMR for IPL, destination mask, and vector if PMCSR_MIE is set.
Miscellaneous Errors		
<30>	NRST	Node reset (WO:ROZ,DCLOC) — Writing a one to this location initiates a complete power-up reset (similar to what happens in response to the assertion and deassertion of XMI#DC#LO#L — see note below); the CIXCD performs self-test and asserts XMI BAD L until it is successfully completed. Just like during power-up reset, other nodes are precluded from accessing the CIXCD from the time NODE RESET has been set until it completes self-test (or the maximum self-test time is exceeded). In response to a real power-up sequence (caused by XMI DC LO L), the NRST bit will be reset; following a NODE RESET sequence, it will remain set. **NOTE** **During the time the CIXCD is responding to NODE RESET, the CIXCD will not access remote nodes on the XMI. In response to a real power-up sequence (caused by XMI DC LO L), this NRST bit will be reset. Following a NODE RESET sequence, NRST will remain set allowing the XMI processor to recognize that it should not attempt to go through the normal boot process.**

Bit	Name	Description
<29>	NHALT	Node halt (R/W:ROZ,DCLOC) — Writing this bit forces the CIXCD to go into a "quiet state" while retaining as much state as possible. When this bit is set, CIXCD commander transactions will be disabled, while responder transactions will complete normally. When this bit is set, RESPCSR_NHALT will also set.
<28>	-	0
<27>	CC	Corrected confirmation (R/W1C:RO,DCLOC) — This bit is set when the node detects a single-bit CNF error (single-bit CNF errors are automatically corrected by the XCLOCK chip in the XMI corner). When set, this bit sets XBER_ES.
<26:24>	-	0
Responder Errors		
<23>	PE	Parity error (R/W1C:RO,DCLOC) — When set, indicates that the CIXCD has detected a parity error on an XMI cycle. The cycle need not have been directed to the CIXCD. When set, this bit sets XBER_ES.
<22>	WSE	Write sequence error (R/W1C:RO,DCLOC) — When set, indicates that the CIXCD aborted a write transaction due to a missing data cycle. When set, this bit sets XBER_ES.
<21>	RIDNAK	Read/IDENT data noack (R/W1C:RO,DCLOC) — When set, indicates that a read data cycle transmitted by the CIXCD has received a noack confirmation. When set, this bit sets XBER_ES.

Commander Errors

<20>	WDNAK	Write data noack (R/W1C:RO,DCLOC) — When set, indicates that a write data cycle transmitted by the CIXCD has received repeated noack confirmations for the duration of the timeout period. Upon receipt of a noack confirmation code on a write data cycle, the CIXCD will retry the entire transaction until it either completes successfully or a TTO (transaction timeout — XBER bit 13) is encountered; in which case this bit will also be set. When set, this bit sets XBER_ES.
<19>	–	0
<18>	NRR	No read response (R/W1C:RO,DCLOC) — When set, indicates that a read or IDENT transaction initiated by the CIXCD failed to receive all of its requested data within the timeout period. If this bit is set, XBER_ES and XBER_TTO will also be set.
<17>	RSE	Read sequence error (R/W1C:RO,DCLOC) — When set, indicates that a read transaction initiated by the CIXCD received its read data out of sequence. When set, this bit sets XBER_ES and the offending command/address will be available in XFADR and XFAER.
<16>	RER	Read error response (R/W1C:RO,DCLOC) — When set, indicates that the CIXCD has received a Read Error Response. When set, this bit sets XBER_ES and the offending command/address will be available in XFADR and XFAER.
<15>	CNAK	Command noack (R/W1C:RO,DCLOC) — When set, indicates that a command cycle transmitted by the CIXCD has received repeated noack confirmations for the entire duration of the timeout period. This can result from a reference to a non-existent memory location or a command cycle parity error. The CIXCD will set this bit when it repeatedly receives a noack confirmation for a given command/address that has been retried for the timeout period. When set, this bit sets XBER_ES and XBER_TTO.
<14>	–	0

Bit	Name	Description
<13>	TTO	Transaction timeout (R/W1C:RO,DCLOC) — When set, indicates that a transaction initiated by the CIXCD has not completed within the timeout period. This bit may set in conjunction with XBER_NRR, XBER_WDNACK, or XBER_CNAK. If none of these bits is set, the CIXCD has either: • Failed to win bus arbitration within the timeout period • Attempted to execute an Iread command, but XMI lockout remained asserted for the timeout period When set, this bit sets XBER_ES and the offending command/address will be available in XFADR and XFAER.

Bit	Name	Description
Node Specific Errors		
<12>	NSES	Node-specific error summary (RO:RO) — This bit is set when a node-specific error condition is detected. The CIXCD sets this bit when one of the following error bits is set in PMCSR: CPDED, CPERR, PRER, PWER, STUF, STOF, YRPE, XRPE, IBPE, CSPE, PRIBPE, MPPBPE, XMRGPE, MAPBPE, MBPBPE, CWXAPE, CWXBPE, MAIBE, MBIBE, CDIBE, ITIBE, RSIBE, MBPIE, MABPE, or MBBPE. The error bit in PMCSR must be cleared in order to clear NSES. When set, this bit sets XBER_ES.
<11>	–	0
<10>	STF	Self-test fail (R/W1C:STS,WO) — While set, this bit indicates that the CIXCD has not yet passed its self-test. The port processor must clear this bit when the CIXCD passes its self-test.
<09:06>	NODEID	Node ID <03:00> (RO:RO) — This field represents the CIXCD's position in the XMI backplane and, therefore, the XMI node ID.
<05:04>	CMDRID	Commander ID <01:00> (RO:R/W,DCLOC) — This field is used to log the commander ID of a failing transaction. When a CMDR, MOVA, MOVB, or INTR XMI fatal error occurs, it is the responsibility of the microcode to load the code of the failing commander in this field. The codes are as follows: 0 = Port xmit mover (mover a) 1 = Port rcv mover (mover B) 2 = Microcode CMDR 3 = INTR
<03>	EHWW	Enable hexword writes (R/W:R/W,DCLOC) — This bit is written by the host during initialization to enable the transmission of hexword writes (hexword write masked) by mover B. If this bit is clear, the maximum write data length will be an octaword.

Bit	Name	Description
<02>	DXTO	Disable XMI timeout (R/W:ROZ,DCLOC) — This bit is used to enable/disable the reporting of all XMI timeouts by the CIXCD. When this bit is set, the CIXCD's internal timeout counter is disabled, preventing any TTO errors. If the CIXCD has a current outstanding XMI transaction when this bit transitions from 0 to 1 (the TTO counters are counting), the given timeout is disabled, and the CIXCD will retry the transaction indefinitely. If the CIXCD has a current outstanding XMI transaction when this bit transitions from 1 to 0 (the TTO counters are not counting), the given timeouts will continue from where they were prior to DXTO being set.
<01>	EMP	Enable more protocol (R/W:R/W,DCLOC) — When set, this bit enables XMOV's data movers to generate read more and write more transactions. More is used only on hexword transfers.
<00>	–	0

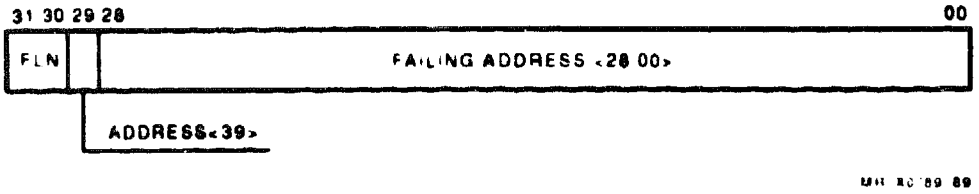


Figure A-3 XMI Failing Address Register LW0 (XFADR) Offset = 00008

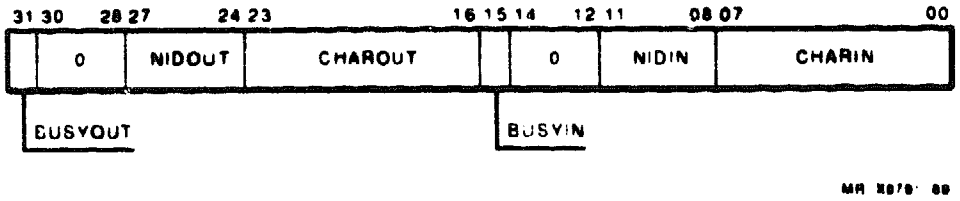


Figure A-4 XMI Communications Register (XCOMM) Offset = 00010

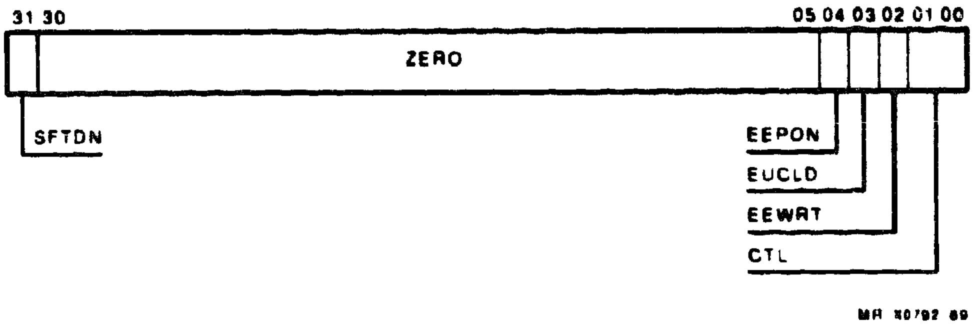


Figure A-5 Port Scan Control Register (PSCR) Offset = 00014

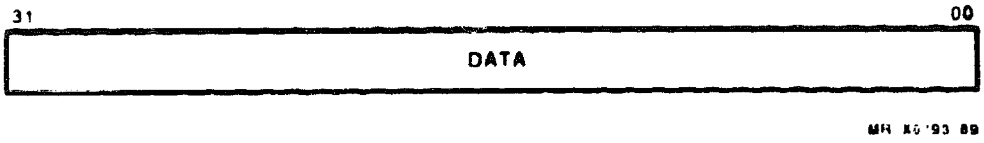


Figure A-6 Port Scan Data Register (PSDR) Offset = 00018

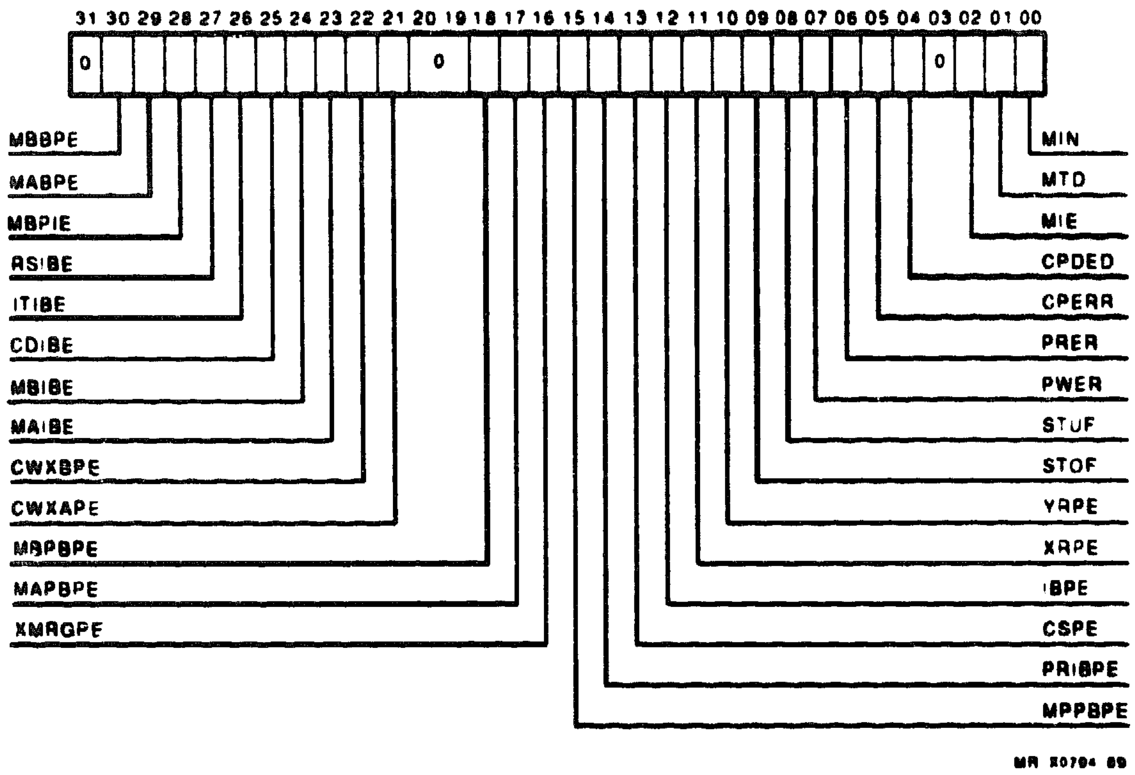


Figure A–7 Port Maintenance Control/Status Register (PMCSR) Offset = 0001C

Bit	Name	Description
<31>	–	0

Error Bits

XMOV Parity Errors

<30>	MBBPE	Mover B byte parity error (R/W1C:R/W,DCLOC) — This bit may be set by the microcode after receiving repeated mover B aborts with MVBCSR_MVBBPE as the error. This allows the microcode the option of retrying the packet before reporting the error to the XMI bus. The setting of this bit also sets XBER_NSES.
<29>	MABPE	Mover A byte parity error (R/W1C:R/W,DCLOC) — This bit may be set by the microcode after receiving repeated mover A aborts with MVACSR_MVABPE as the error. This allows the microcode the option of retrying the packet before reporting the error to the XMI bus. The setting of this bit also sets XBER_NSES.
<28>	MBPIE	MOVB detected PB_IB parity error on PB read (R/W1C:R/W,DCLOC) — This bit may be set by the microcode after receiving repeated mover B aborts with MVBCSR_MVBPBE as the error. This allows the microcode the option of retrying the packet before reporting the error to the XMI bus. The setting of this bit also sets XBER_NSES.
<27>	RSIBE	RESP PB_IB parity error on reg write (R/W1C:R/W,DCLOC) — This bit, when set, indicates that a parity error was detected over the PB_OR_IB bus on a register write to the responder. The setting of this bit also sets XBER_NSES.
<26>	ITIBE	INTR PB_IB parity error on reg write (R/W1C:R/W,DCLOC) — This bit, when set, indicates that a parity error was detected over the PB_OR_IB bus on a register write to the interrupt controller. The setting of this bit also sets XBER_NSES.

Bit	Name	Description
<25>	CDIBE	CMDR PB_IB parity error on reg write (R/W1C:R/W,DCLOC) — This bit, when set, indicates that a parity error was detected over the PB_OR_IB bus on a register write to the commander. The setting of this bit also sets XBER_NSES.
<24>	MBIBE	MOVB PB_IB parity error on reg write (R/W1C:R/W,DCLOC) — This bit, when set, indicates that a parity error was detected over the PB_OR_IB bus on a register write to mover B. The setting of this bit also sets XBER_NSES.
<23>	MAIBE	MOVA PB_IB parity error on reg write (R/W1C:R/W,DCLOC) — This bit, when set, indicates that a parity error was detected over the PB_OR_IB bus on a register write to mover A. The setting of this bit also sets XBER_NSES.

Bit	Name	Description
MCWI Parity Errors		
<22>	CWXBPE	CWIN transmit path B parity error (R/W1C:R/W,DCLOC) — This bit is set by the microcode when the CI wire interface logic, during a transmit function on path B, detects bad parity from either the byte wide transmit data path leaving the MCWI to the CI corner logic or the conversion of longword PB data to byte-wide transmit data in the MCWI. The setting of this bit also sets XBER_NSES.
<21>	CWXAPE	CWIN transmit path A parity error (R/W1C:R/W,DCLOC) — This bit is set by the microcode when the CI wire interface logic, during a transmit function on path A, detects bad parity from either the byte-wide transmit data path leaving the MCWI to the CI corner logic or the conversion of longword PB data to byte-wide transmit data in the MCWI. The setting of this bit also sets XBER_NSES.
<20:19>	–	0
<18>	MBPBPE	Mover B packet buffer read parity error (R/W1C:R/W,DCLOC) — The memory controller sets this bit to indicate that bad parity was detected on mover B PB read data received by the memory controller from the PB RAMs over the MCWI_PB data bus (PB memory bus).
<17>	MAPBPE	Mover A packet buffer write parity error (R/W1C:R/W,DCLOC) — The memory controller sets this bit to indicate that bad parity was detected on mover A packet buffer write data received by the memory controller from the XMOV gate arrays over the PB_OR_IB data bus.

Bit	Name	Description
<16>	XMRGPE	XMOV register read parity error (R/W1C:R/W,DCLOC) — The memory controller sets this bit to indicate that bad parity was detected on XMOV register read data received by the memory controller from the XMOV gate array over the PB_OR_IB data bus. The destination of this XMOV register read data is the MCDP microcontrol and data path gate array.
<15>	MPPBPE	MCDP packet buffer read parity error (R/W1C:R/W,DCLOC) — The memory controller sets this bit to indicate that bad parity was detected on MCDP packet buffer read data received by the memory controller from the PB RAMs over the MCWI_PB data bus (PB memory bus).
<14>	PRIBPE	PORT_IB receive parity error (R/W1C:R/W,DCLOC) — The memory controller sets this bit to indicate that bad parity was detected on data received by the memory controller from the MCDP gate array over the PORT IB.
MCDP Parity Errors		
<13>	CSPE	Control store parity error (R/W1C:R/W,DCLOC) — This bit is set by the microcode when the port processor encounters a control store parity error. This bit can only be set if the microcode can recover enough to write this bit. This bit must be written by the port processor when the error occurs. The setting of this bit also sets XBER_NSES.
<12>	IBPE	Internal bus parity error (R/W1C:R/W,DCLOC) — This bit is set when the port processor encounters an internal bus parity error. This bit can only be set if the microcode can recover enough to write this bit. This bit must be written by the port processor when the error occurs. The setting of this bit also sets XBER_NSES.

Bit	Name	Description
<11>	XRPE	X register parity error (R/W1C:R/W,DCLOC) — This bit is set when the parity of the X register in the port processor data path is incorrect. This bit can only be set if the microcode can recover enough to write this bit. This bit must be written by the port processor when the error occurs. The setting of this bit also sets XBER_NSES.
<10>	YRPE	Y register parity error (R/W1C:R/W,DCLOC) — This bit is set when the parity of the Y register in the port processor data path is incorrect. This bit can only be set if the microcode can recover enough to write this bit. This bit must be written by the port processor when the error occurs. The setting of this bit also sets XBER_NSES.
<09>	STOF	Microstack overflow (R/W1C:R/W,DCLOC) — This bit is set when the microstack is overflowed by too many pushes. This bit can only be set if the microcode can recover enough to write this bit. This bit must be written by the port processor when the error occurs. The setting of this bit also sets XBER_NSES.
<08>	STUF	Microstack underflow (R/W1C:R/W,DCLOC) — This bit is set when the microstack is underflowed by too many pops. This bit can only be set if the microcode can recover enough to write this bit. This bit must be written by the port processor when the error occurs. The setting of this bit also sets XBER_NSES.
Port Errors		
<07>	PWER	Port write error response (R/W1C:R/W,DCLOC) — This bit is set as a result of the microcode having set INTCTR_SWEI. INTCTR_SWEI is set by the microcode when it wishes to send a write error IVINTR, if the CIXCD is unable to complete a register write transaction to its node space but not to a valid register.

Bit	Name	Description
<06>	PRER	Port read error response (R/W1C:R/W,DCLOC) — This bit is set as a result of the microcode having set RESPCSR_SNDRER. RESPCSR_SNDRER is set by the microcode when it wishes to send an RER response to a register read, if the read is within its node space but not to a valid register. Since the read error response caused a machine check, an interrupt is *not* generated.
<05>	CPERR	CP_ERROR_STATUS (R/W1C:RO,DCLOC) — This bit is set if the CP_ERROR_STATUS signal remains asserted for more than 32 cycles. CP_ ERROR_STATUS is set when any of the bits in the MCDP internal conditions register are set. When one of these errors occurs, the microcode will trap and execute a port shutdown routine, which clears CP_ERROR_STATUS, if the failure was intermittent. If this bit is set, all other MCDP error bits in this register are not valid; this is the only indication that the port processor has had an unrecoverable failure. Scan data can provide additional data. The setting of this bit also sets XBER_NSES.
<04>	CPDED	CPU no response error (R/W1C:RO,DCLOC) — This bit is set if the port processor fails to respond to a responder interrupt within 512 cycles. The port processor is assumed to have failed. The setting of this bit also sets XBER_NSES.

Bit	Name	Description
Control Bits		
<02>	MIE	Maintenance interrupt enable (R/W:RO,DCLOC) — This bit, when set, enables XMI interrupts.
<01>	MTD	Maintenance/sanity timer disable (R/W:RO,DCLOC) — If set, the maintenance sanity/timer is set to the initial value and suspended. If clear, the timer functions normally. This bit resides as a flip-flop in XMOV hardware, which sets or clears when this bit is written. On PMCSR reads from the XMI bus, the state of this flip-flop is returned as MTD; on PMCSR writes from the XMI bus, the register write is sent to the port processor as a virtual write, and the local MTD flip-flop is updated as well.
<00>	MIN	Maintenance init (WO:RO,DCLOC) — When set, clears all hardware state including errors and puts the port in the uninitialized state, without copying microcode from EEPROM to RAM or executing self-test

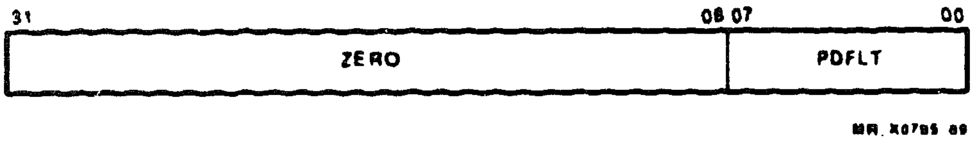


Figure A-8 Port Diagnostic Control/Status Register (PDCSR) Offset = 00020

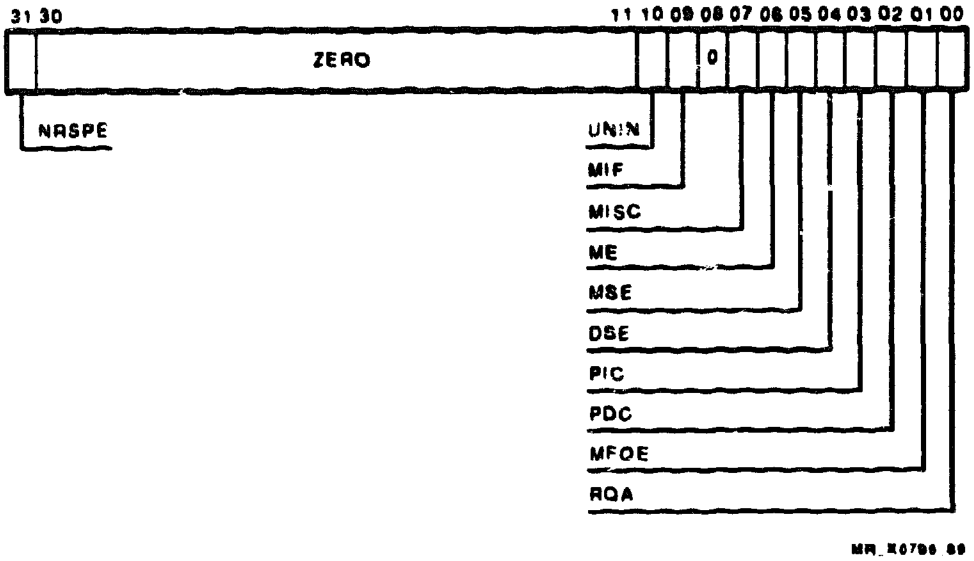


Figure A-9 Port Status Register (PSR) Offset = 00024

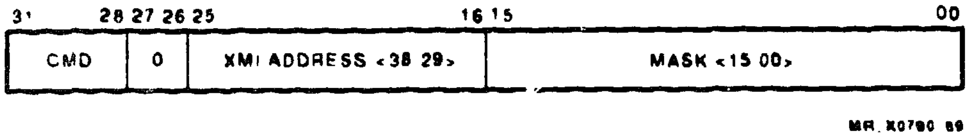


Figure A-10 XMI Failing Address Register LW1 (XFAER) Offset = 0002C

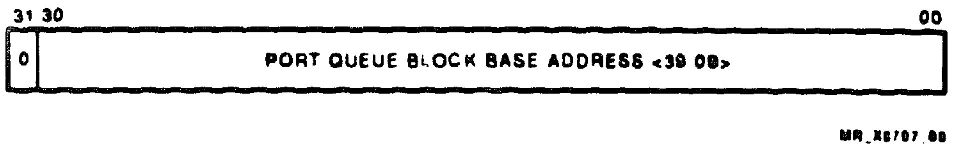


Figure A–11 Port Queue Block Base Register (PQBBR) Offset = 01000

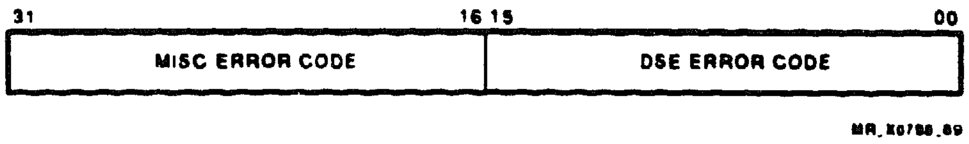


Figure A–12 Port Error Status Register (PESR) Offset = 01008

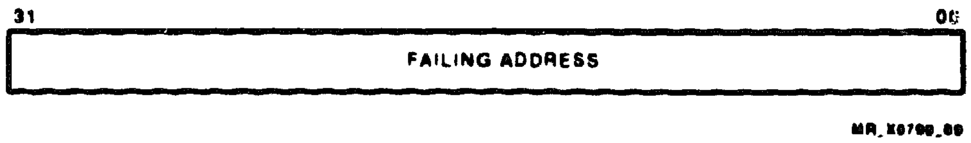


Figure A–13 Port Failing Address Register (PFAR) Offset = 0100C

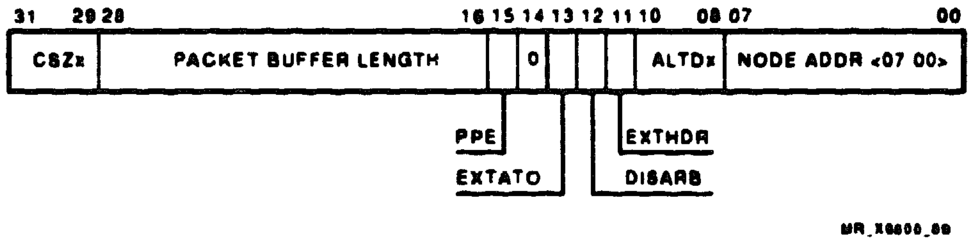


Figure A–14 Port Parameter Register (PPR) Offset = 01010

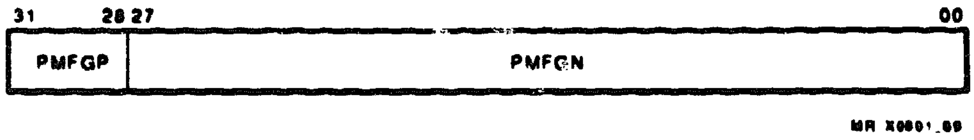


Figure A–15 Port Serial Number Register (PSNR) Offset = 01014

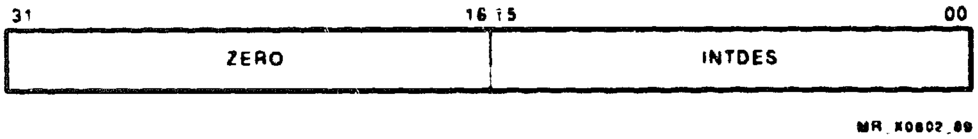


Figure A–16 Port Interrupt Destination Register (PIDR) Offset = 01018

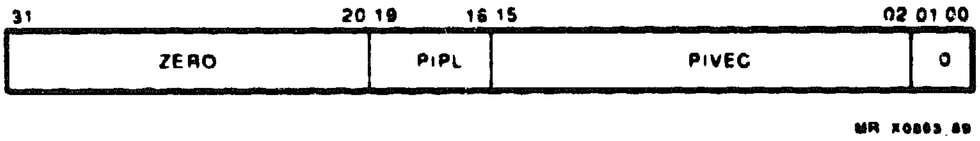


Figure A–17 Port Interrupt Vector Register (PIVR) Offset = 01020

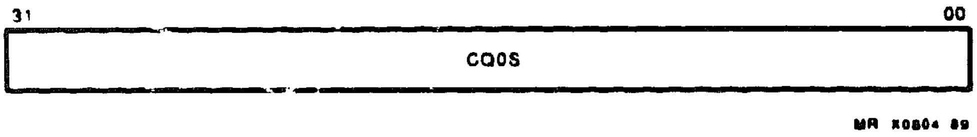


Figure A–18 Port Command Queue 0 Control Register (PCQ0CR) Offset = 01028

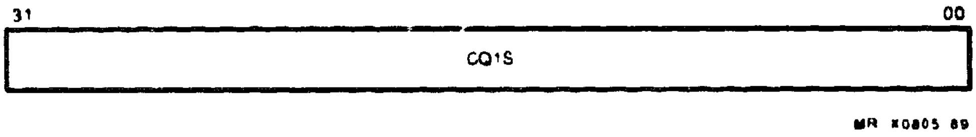


Figure A–19 Port Command Queue 1 Control Register (PCQ1CR) Offset = 0102C

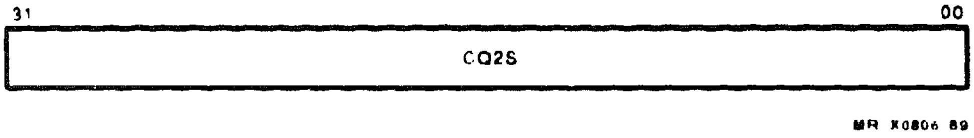


Figure A–20 Port Command Queue 2 Control Register (PCQ2CR) Offset = 01030

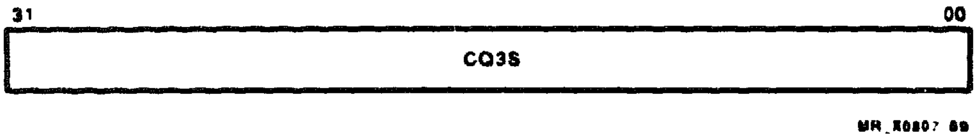


Figure A–21 Port Command Queue 3 Control Register (PCQ3CR) Offset = 01034

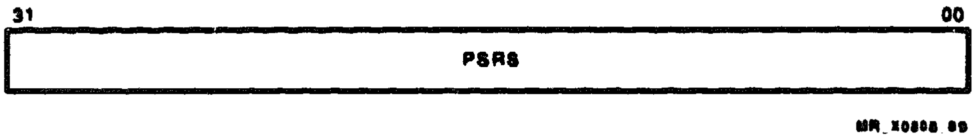


Figure A–22 Port Status Release Control Register (PSRCR) Offset = 01038

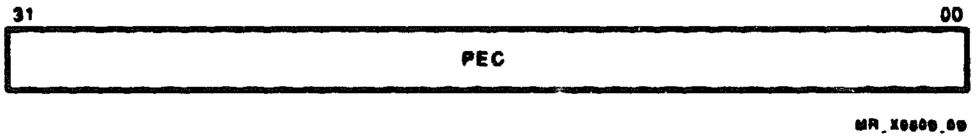


Figure A–23 Port Enable Control Register (PECR) Offset = 0103C

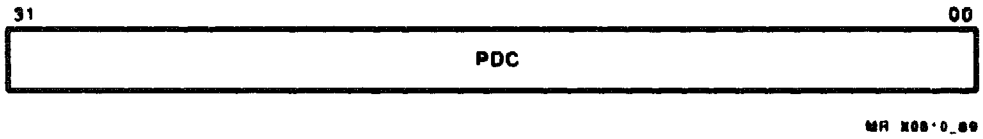


Figure A–24 Port Disable Control Register (PDCR) Offset = 01040

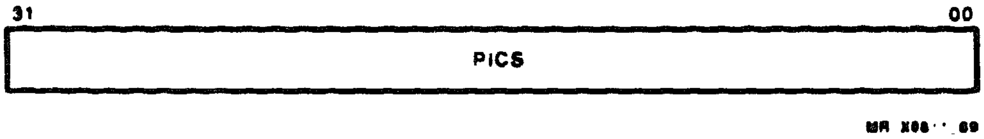


Figure A–25 Port Initialize Control Register (PICR) Offset = 01044

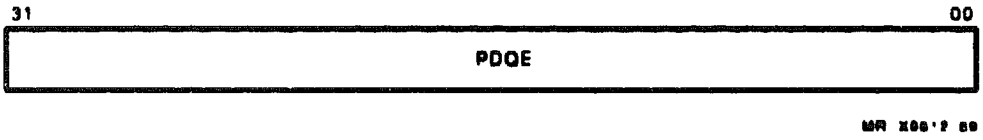


Figure A–26 Port Datagram Free Queue Control Register (PDFQCR) Offset = 01048

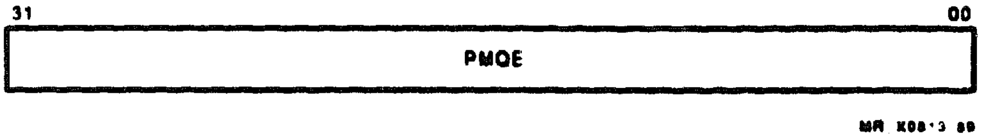


Figure A–27 Port Message Free Queue Control Register (PMFQCR) Offset = 0104C

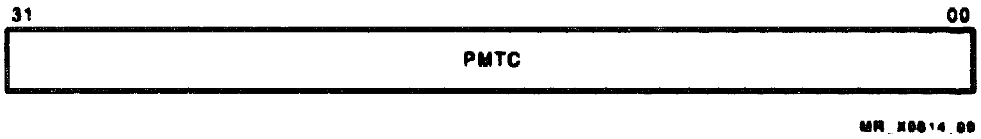


Figure A–28 Port Maintenance/Sanity Timer Control Register (PMTCR) Offset = 01050

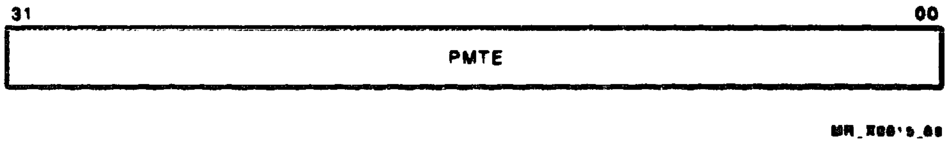


Figure A–29 Port Maintenance/Sanity Timer Expiration Control Register (PMTECR) Offset = 01054

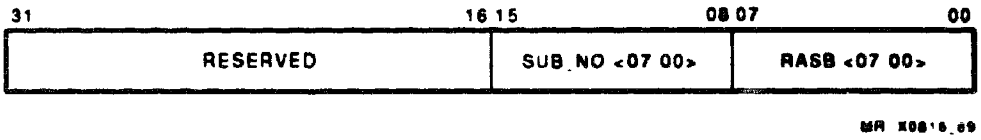


Figure A–30 Port Parameter Extension Register (PPER) Offset=01058

APPENDIX B

B EVGEA Sections

```
DS>
DS> start/section=update

.. Program: EVGEA CIXCD Functional Level 3 Diag , revision 2.0, 25
 tests,
   at 14:04:25.68.
Testing: _PAB0

 CIXCD EEPROM update utility

The current CIXCD XMI node number is 0C

Input the CIXCD BINARY filename to use [default= CIXCD.BIN]: RETURN

  Loading Microcode file CIXCD.BIN
<------------- start of Microcode file header text block ---------->


Copyright (c) Digital Equipment Corporation 1989. All rights reserved.

Diagnostic Firmware Revision 1.0 , Functional Firmware Revision 1.04


<--------------- end of Microcode file header text block ---------->

Starting to write EEPROM Diagnostic BACKUP
Starting to write EEPROM Diagnostic PRIMARY
Starting to write EEPROM Functional BACKUP
Starting to write EEPROM Functional PRIMARY
Writting EEPROM has been completed

Number of EEPROM write cycles = 0001


.. End of run, 0 errors detected, pass count is 1,
   time is 27-MAR-1990 15:59:56.75
DS>
```

Example B–1 UPDATE Section

```
DS>
DS> start/section=verify

.. Program: EVGEA CIXCD Functional Level 3 Diag , revision 2.0, 25
 tests,
   at 14:06:42.06.
Testing:  PABC

 CIXCD EEPROM verify utility

The current CIXCD XMI node number is 0C
Input the CIXCD BINARY filename to use [default= CIXCD.BIN]: RETURN

  Microcode file already loaded in VAX ram memory

<------------- start of Microcode file header text block ---------->


Copyright (c) Digital Equipment Corporation 1989. All rights reserved.

Diagnostic Firmware Revision 1.0 , Functional Firmware Revision 1.04


<--------------- end of Microcode file header text block ---------->


Verifying EEPROM has been completed

.. End of run, 0 errors detected, pass count is 1,
   time is 30-MAR-1990 14:06:55.34
DS>
```

Example B-2 VERIFY Section, No Errors

```
DS>
DS> start/section=verify

.. Program: EVGEA CIXCD Functional Level 3 Diag , revision 2.0, 25
 tests,
   at 14:09:41.82.
Testing: _PABC

 CIXCD EEPROM verify utility

The current CIXCD XMI node number is 0C
Input the CIXCD BINARY filename to use (default= CIXCD.BIN): [RETURN]

  Microcode file already loaded in VAX ram memory

<------------- start of Microcode file header text block ---------->


Copyright (c) Digital Equipment Corporation 1989. All rights reserved.

Diagnostic Firmware Revision 1.0 , Functional Firmware Revision 1.04

<--------------- end of Microcode file header text block ---------->

********  EVGEA CIXCD Functional Level 3 Diag  - 1.0  ********
 Pass 1, test 13, subtest 0, error 6, 30-MAR-1990 14:09:50.52
 Hard error while testing PABC: EEPROM region data did not VERIFY
  correctly

  Xored CS EEPROM = 00001000 00000000 0000C002
Reading CS EEPROM = 0000C000 01000110 00000000
Verify ucode file = 0000D000 01000110 0000C002, Loc 1234

********  End of Hard error number 6 ********

There were 1. EEPROM addresses in error
composite of all CS EEPROM XOR'd bits in error - 001000 00000000 0000C002

Module H/W revision- = D02
Chip fault data        00  10  00  00  00  00  00  00  00  00  02
Chip fault list        ----E14-----------------------------E50-E61

Verifying EEPROM has been completed

.. End of run, 1 error detected, pass count is 1,
   time is 30-MAR-1990 14:10:01.54
DS>
```

Example B-3 VERIFY Section, Errors

```
DS>
DS> start/section=rverify
.. Program: EVGEA CIXCD Functional Level 3 Diag , revision 2.0, 25
 tests,
   at 14:23:08.47.
Testing: _PAB0
 CIXCD EEPROM Region-Verify EEPROM utility
The current CIXCD XMI node number is 0C
Starting to verify EEPROM Diagnostic PRIMARY->BACKUP
There were 0. Diagnostic region EEPROM locations different
Starting to verify EEPROM Functional PRIMARY->BACKUP
There were 0. Functional region EEPROM locations different
Verifying EEPROM has been completed
.. End of run, 0 errors detected, pass count is 1,
   time is 30-MAR-1990 14:23:17.78
DS> st/se=exam
DS>
```

Example B-4 RVERIFY Section, No Errors

```
DS>
DS> start/section=rverify

.. Program: EVGEA CIXCD Functional Level 3 Diag , revision 2.0, 25
 tests,
   at 14:19:01.83.
Testing: _PAB0

 CIXCD EEPROM Region-Verify EEPROM utility

The current CIXCD XMI node number is 0C

Starting to verify EEPROM Diagnostic PRIMARY->BACKUP
********  EVGEA CIXCD Functional Level 3 Diag  - 1.0  ********
 Pass 1, test 19, subtest 0, error 65535, 30-MAR-1990 14:19:05.86
 Hard error while testing PAB0: EEPROM region data did not compare
  correctly

 Backup region = 0000C000 01000110 00000000, Address = 1234
Primary region = 0000D000 01000110 0000C002, Address = 3234

********  End of Hard error number 65535 ********


There were 1. Diagnostic region EEPROM locations different

Starting to verify EEPROM Functional PRIMARY->BACKUP
There were 0. Functional region EEPROM locations different

Verifying EEPROM has been completed

.. End of run, 1 error detected, pass count is 1,
   time is 30-MAR-1990 14:19:14.90
DS>
```

Example B-3 RVERIFY Section, Errors

```
DS>
DS> start/section=replace
.. Program: EVGEA CIXCD Functional Level 3 Diag , revision 2.0, 25
 tests,
   at 14:22:32.36.
Testing: _PAB0
 CIXCD EEPROM replace EEPROM utility
The current CIXCD XMI node number is 0C

Starting to write EEPROM Diagnostic BACKUP
There were 1. Diagnostic region EEPROM locations written
Starting to write EEPROM Functional BACKUP
There were 0. Functional region EEPROM locations written
Writting EEPROM has been completed

.. End of run, 0 errors detected, pass count is 1,
   time is 30-MAR-1990 14:22:42.86
DS>
```

Example B-6 REPLACE Section

```
DS>
DS> start/section=restore
.. Program: EVGEA CIXCD Functional Level 3 Diag , revision 2.0, 25
 tests,
   at 14:25:43.24.
Testing: _PAB0
 CIXCD EEPROM restore EEPROM utility
The current CIXCD XMI node number is 0C

Starting to write EEPROM Diagnostic PRIMARY
There were 0. Diagnostic region EEPROM locations written
Starting to write EEPROM Functional PRIMARY
There were 0. Functional region EEPROM locations written
Writting EEPROM has been completed

.. End of run, 0 errors detected, pass count is 1,
   time is 30-MAR-1990 14:25:53.74
DS>
```

Example B–7 RESTORE Section

```
DS>
DS> start/section=errorlog
.. Program: EVGEA CIXCD Functional Level 3 Diag , revision 2.0, 25
 tests,
   at 14:01:41.31.
Testing: _PAB0
 CIXCD EEPROM examine DCB-ERRORLOG  utility
The current CIXCD XMI node number is 0C

  The current date and time is = 30-MAR-1990 14:01:43.84
EEPROM UPDATE date and time is = 27-MAR-1990 15:59:56.75
Values currently read from EEPROM
Module Serial Number = AS00400190
Module H/W revision- = E03
Functional Microcode revision- - - - = 1.04
This version of microcode supports 1K Packet buffer size
Diagnostic Microcode revision- - - - = 1.0
Functional Microcode region Checksum = E75B631B
Diagnostic Microcode region Checksum = C2442F6B
Diagnostic Control Block Checksum- - = 00000000
Number of valid ERRORLOG entries = 08
Number of EEPROM write cycles = 0C01


.. End of run, 0 errors detected, pass count is 1,
   time is 30-MAR-1990 14:01:50.02
DS>
```

Example B-8 ERRORLOG Section

```
DS>
DS> START/SECTION=EXAM

.. Program: EVGEA CIXCD Functional Level 3 Diag , revision 2.0, 25
 tests,
   at 14:17:22.93.
Testing: _PAB0

 CIXCD EEPROM examine ERRORLOG entry utility

The current CIXCD XMI node number is 0C

Number of valid ERRORLOG entries = 08

Examine error entry # (RETURN = All) [(0000), 0001-0008(X)]  1 RETURN

Error Log entry 01:
        Failing Test Number- - - 13, Subtest  0
        Failing Address- - - - - 00001234
        Expected Data- - - - - - 0000D000  01000110  0000C002
        Actual Data- - - - - - - 0000C000  01000110  00000000
        Date and Time of Error - 30-MAR-1990 14:09:50.50

.. End of run, 0 errors detected, pass count is 1,
   time is 30-MAR-1990 14:17:34.14
DS>
```

Example B–9 EXAMLOG Section

APPENDIX C

C
Cluster Upgrades

C.1 CIXCD VAXcluster Minimum Revision Levels

Table C–1 lists minimum revision levels.

Table C–1 Minimum Revision Levels

	Environment	
Device	1-16 Nodes (SC008)	1-32 Nodes (CISCE)
VMS	V5.4	V5.4
CIXCD		
T2080	Rev E02	Rev E02
CIXCD.BIN	V1.0	V1.0
HSC		
Link module	L0100 rev E or L0118 rev B	L0118 rev B
CRONIC:[1]		
HSC50	V400	V400
HSC40/70	V500	V500
CI7x0, CIBCI		
Link module	L0100 rev E or L0118 rev B	L0118 rev B
CI780.BIN/L0101[2]	Rev 8.7/L0101 rev K or Rev 20.20/L0101-YA	Rev 20.20/L0101-YA
CIBCA-A		
CIBCA.BIN[3]	Rev 7.5	Rev 7.5
CIBCA-B		
CIBCB.BIN[4]	Rev 5.2	Rev 5.2

[1]CRONIC and microcode revision levels can be examined using the SHOW CLUSTER utility with the ADD RP_REV command.

[2]CI780.BIN is displayed as either 80007 or 200020, which equate to 8.7 and 20.20.

[3]CIBCA.BIN is displayed as 70005, which equates to 7.5.

[4]CIBCB.BIN is displayed as 40054002, which equates to 5.2.

C.2 Quiet Slot Time Settings

A quiet slot time of seven (7-tick mode) does not ensure correct propogation time to avoid excessive collisions during arbitration. It is recommended that all nodes be set to have the quiet slot time set to 10 (10-tick mode) regardless of the environment. Clusters must not have individual nodes running in mixed mode operation (that is, some 7-tick mode and others in 10-tick mode).

Ten-tick mode is *required* in all clusters containing a CIXCD or otherwise having high-traffic applications.

If a cluster is running with nodes operating in both 7-tick mode and 10-tick mode, then all nodes *must* be changed to 10-tick mode. For the recommended procedure for upgrading a node from 7-tick mode to 10-tick mode, refer to the following steps:

1. Examine all nodes before making any changes to the slot times.
2. If changes are required, shut down *all* nodes *not* currently in 10-tick mode before making any changes.
3. Change all nodes currently in 7-tick mode to 10-tick mode using the following procedures:
   - CIXCD-AA, CIXCD-AB — Put a jumper (12-14314-01) into jumper position W30 (E15/45, refer to Section 3.5.5) of the slot in the XMI backplane containing the T2080 module. Ensure that W28 (E13/43) and W29 (E14/44) do not have jumpers.
   - CIBCA-AA, CIBCA-BA — Put a jumper (12-14314-01) into jumper position E11/41 (refer to Figure 3–4) of the slot in the VAXBI backplane that contains the T1015/T1045 module. Ensure that E09/39 and E10/40 do not have jumpers.
   - CI780, CI750, CIBCI, all HSCs — Change the required switch settings as follows:
     — For devices with L0100 modules: Place both elements of S3 in the ON position. S3 is a two-segment DIP switch located adjacent to S1 and S2.

       NOTE
       All L0100 rev D modules must be replaced with either L0100 rev E or L0118 rev B modules.

- For devices with L0118 modules: Place switch elements 2 and 3 in the off position and element 4 in the on position. S3 is a four-segment DIP switch located adjacent to S1 and S2. (Element 1 of S3 must be turned on in VAXcluster configurations utilizing the CISCE star coupler expander, or it must be turned off in all other star coupler configurations.

4. Reboot node after change is complete.

INDEX

Index