DECmpp 12000/Sx Model 100

Hardware Service Manual

Part Number: EK–DECAC–SM. C01

September 1992

This document provides service and diagnostic procedures for DECmpp 12000/Sx and DECmpp 12000–LC/Sx Series systems.

Revision/Update Information:	This document has been revised for DECmpp Version 1.1.
Operating System and Version:	ULTRIX Version 4.2A. Future releases may require higher versions.
Software Version:	DECmpp 12000/Sx Version 1.1.

Digital Equipment Corporation
Maynard, Massachusetts

First Printing, January 1992
Revised, September 1992

The information in this document is subject to change without notice and should not be construed as a commitment by Digital Equipment Corporation. Digital Equipment Corporation assumes no responsibility for any errors that may appear in this document.

The software described in this document is furnished under a license and may be used or copied only in accordance with the terms of such license.

NOTICE—Class A Computing Device:
This equipment generates, uses, and may emit radio frequency energy. The equipment has been type tested and found to comply with the limits for a Class A computing device pursuant to Subpart J or Part 15 of FCC rules, which are designed to provide reasonable protection against such radio frequency interference when operated in a commercial environment. Operation of this equipment in a residential area may cause interference; in which case, measures taken to correct the interference are at the user's expense.

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

© Digital Equipment Corporation 1992.

All Rights Reserved.

The postpaid Reader's Comments forms at the end of this document request your critical evaluation to assist in preparing future documentation.

The following are trademarks of Digital Equipment Corporation: DECnet, DECstation, DECsupport, DECsystem, DECwindows, Rdb/VMS, ThinWire, TURBOchannel, ULTRIX, VAX, VAX DOCUMENT, VMS, and the DIGITAL logo.

The following are registered trademarks of the MasPar Computer Corporation: MasPar and the MasPar logo. The following are trademarks of the MasPar Computer Corporation: MasPar Data Display Library (MPDDL), MasPar FORTRAN (MPF), MasPar Input/Output Channel, MasPar Parallel Application Language (MPL), MasPar Parallel Disk Array (MPDA), and MasPar Programming Environment (MPPE). UNIX is a registered trademark of UNIX System Laboratories, Inc.

This document was prepared using VAX DOCUMENT, Version 2.0.

Contents

6 Removal and Replacement Procedures

7 Backplane Jumpers and Upgrading PE Arrays

A Recommended Spares List

B Data Parallel Unit Reference Pages

Index

Figures

Tables

Preface

This manual provides service and diagnostic procedures for DECmpp 12000/Sx systems. Anyone who services DECmpp 12000/Sx Series systems or needs to replace any items should read this manual and be familiar with the procedures.

Intended Audience

This guide is for use by Digital Services personnel and by self-maintenance customers who will be servicing the DECmpp 12000 and DECmpp 12000–LC systems.

Document Structure

The *DECmpp 12000/Sx Hardware Service Manual* contains seven chapters and two appendixes.

- Chapter 1 is a system overview and contains the power-up and power-down routines for both the data parallel unit (DPU) and the front-end server.
- Chapter 2 describes the DPU switches, controls, indicators, and the DPU power system.
- Chapter 3 describes the DPU power supply settings and how to adjust them, if necessary.
- Chapter 4 describes the DECmpp 12000/Sx system cables, connectors, and the printed circuit boards (PCBs) outside the DPU card cage.
- Chapter 5 describes the diagnostic software provided, how to use it, and how to interpret the results.
- Chapter 6 explains the procedures for removing and replacing the DPU components.
- Chapter 7 explains the procedures for upgrading the DECmpp 12000/Sx Series systems by adding more processor element (PE) array PCBs.
- Appendix A lists all of the recommended spare parts for the DPU.
- Appendix B is a collection of Reference Pages that apply to the DPU.

Related Documents

Table 1 lists documents that provide additional information about the DECmpp 12000/Sx system.

Table 1 Related Documents

Document Title	Order Number
DECmpp 12000/Sx System Overview Manual	AA-PMAPB-TE
DECmpp 12000/Sx System Administration Guide	AA-PKU3C-TE
DECmpp 12000/Sx Architecture Specification	AA-PMASB-TE
DECmpp 12000/Sx Parallel Disk Array Reference Manual	EK-DECAB-RM
DECmpp 12000/Sx Parallel VME Reference Manual	EK-DECAB-PM
DECmpp 12000/Sx Hardware Installation Guide	EK-DECAC-IG
DECstation 5000/240 User Documentation Kit	EK-PM380-DK
DECstation 5000/240 Maintenance Guide	EK-PM38C-MG
DECstation 5000/240 Pocket Service Guide	EK-PM38D-PG
DECsystem 5900 Site Preparation Guide	EK-D590A-SP
DECsystem 5900 Installation Guide	EK-D590A-IN
DECsystem 5900 Owner's Guide	EK-D590A-OG
DECsystem 5900 Pocket Service Guide	EK-D590A-PS
DECsystem 5900 Enclosure Maintenance Manual	EK-D590A-EN
DWTVX-Ax VME I/O Subsystem Pocket Service Guide	EK-DWTVX-PS
T6000 Module Installation/Owner's Card	EK-T6000-IN

Conventions

The following conventions are used throughout the DECmpp 12000/Sx documentation set:

Convention	Meaning
Return	In examples, a key name shown within a box indicates that you press a key on the keyboard. In text, a key name is not enclosed in a box but is printed with an initial capital letter, like Return.
Ctrl/*x*	A key combination, shown with a slash separating the two key names, indicates that you hold down the first key while you press the second key.
MB1, MB2, MB3	The buttons on a mouse. MB1 is the left button, MB2 is the center button, and MB3 is the right button of a mouse whose button arrangement is right-handed. It is possible to redefine the mouse buttons.
%	A percent sign (%) represents the default user prompt for your system.
#	A number sign (#) represents the default superuser (root) prompt for your system.
. . .	In examples, a horizontal series of dots, or ellipsis, indicates that additional parameters, values, or other information can be entered.
⋮	In examples, a vertical series of dots, or vertical ellipsis, indicates that a portion of the example is intentionally omitted.
[ ]	In syntax descriptions and functional descriptions, brackets indicate optional items.
`dpumanager(6)`	Cross-references to the *ULTRIX Reference Pages*, which include the appropriate section number in parentheses.
italicized text	In examples, italicized text denotes parameters, values, or other information that will change from either session to session or user to user. In text, italicized words or phrases are used to add emphasis to important words, concepts, or titles of manuals.
`ULTRIX keywords`	This typeface is used to indicate system output or the exact name of a command, option, partition, pathname, directory, or file.
`Code examples`	This typeface is used to display program coding examples.
UPPERCASE and lowercase strings	The ULTRIX system differentiates between lowercase and uppercase characters. Literal strings that appear in text, examples, syntax descriptions, and function descriptions must be entered exactly as shown.

Both the DECmpp 12000/Sx and DECmpp 12000–LC/Sx Series hardware systems are described in this manual. However, because the two systems are very similar, references to the DECmpp 12000 system also apply to the DECmpp 12000–LC system, unless specific differences between the two systems are noted.

Three types of notes are used in this manual:

- **Note**

  Gives additional information or information particularly important to the procedure.

- **Caution**

  Indicates potential damage to equipment or data.

- **Warning**

  Indicates potential injury to a person.

1

System Overview

DECmpp 12000/Sx Series systems are powerful single-instruction, multiple data (SIMD) computers, consisting of a data parallel unit (DPU), which performs the parallel calculations, and a front-end server. The server runs the ULTRIX operating system and functions as a scalar processor. A high-speed VMEbus interface carries data between the DPU and the front-end server.

The DPU contains the array control unit (ACU) PCB and from 1 to 16 processor element (PE) array PCBs. The ACU controls the PE arrays, which perform the parallel calculations. Each PE array PCB provides 1024 processor elements. The total collection of PE array PCBs in a system comprise the PE array.

DECmpp 12000 systems support 1, 2, 4, 8, or 16 PE array PCBs and have 15 I/O slots for optional I/O PCBs. The DPU is housed within an H9A00 series enclosure. DECmpp 12000 systems may be ordered with either a DECsystem 5900 server or a DECstation 5000/240 server. The DECsystem 5900 is contained within a separate H9A00 series enclosure. This configuration is illustrated in Figure 1–1. The DECstation 5000/240 and its storage devices are contained within two desktop boxes. This configuration is illustrated in Figure 1–2.

DECmpp 12000–LC systems support 1, 2, or 4 PE array PCBs and have 5 I/O slots for optional I/O PCBs. The DPU is contained within a low profile cabinet and connects to a table-top DECstation 5000 server (Figure 1–3).

Any of the DECmpp 12000/Sx/Sx system configurations may also contain an optional Parallel Disk Array (PDA). The PDA subsystem is housed in a separate H9A00 series enclosure.

Figure 1–1 Typical DECmpp 12000/Sx Installation with DECsystem 5900 Server

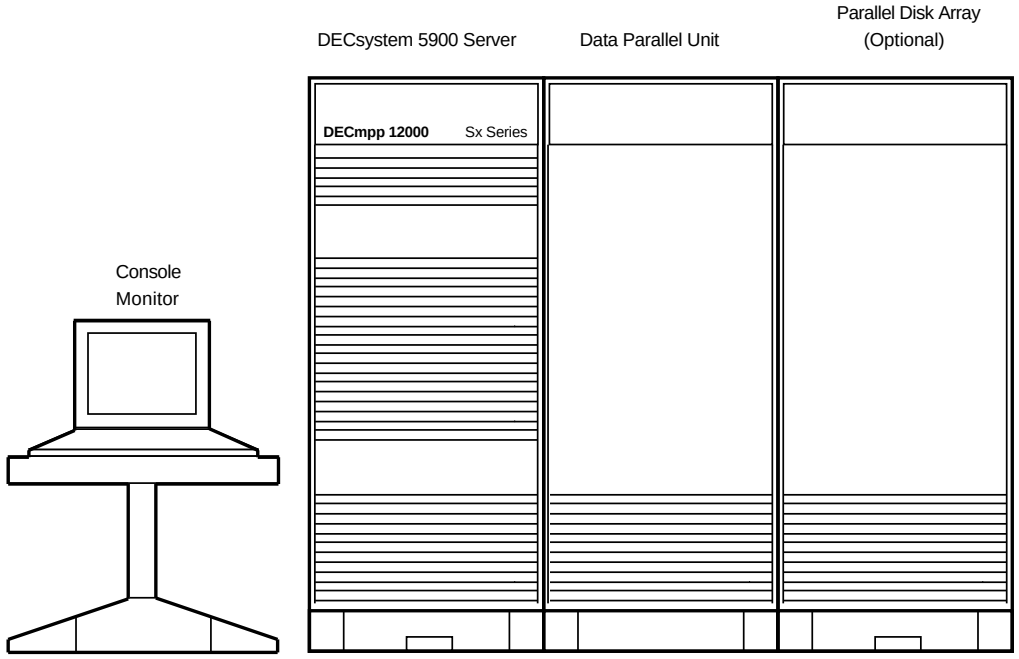


Figure 1–2 Typical DECmpp 12000/Sx Installation with DECstation 5000/240 Server

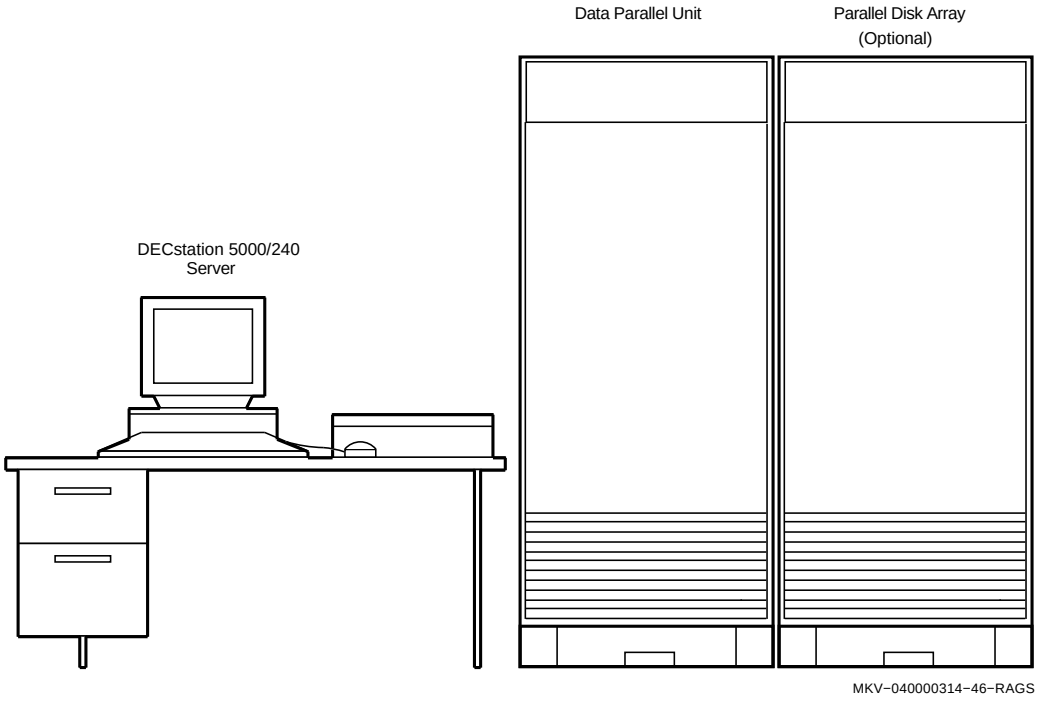
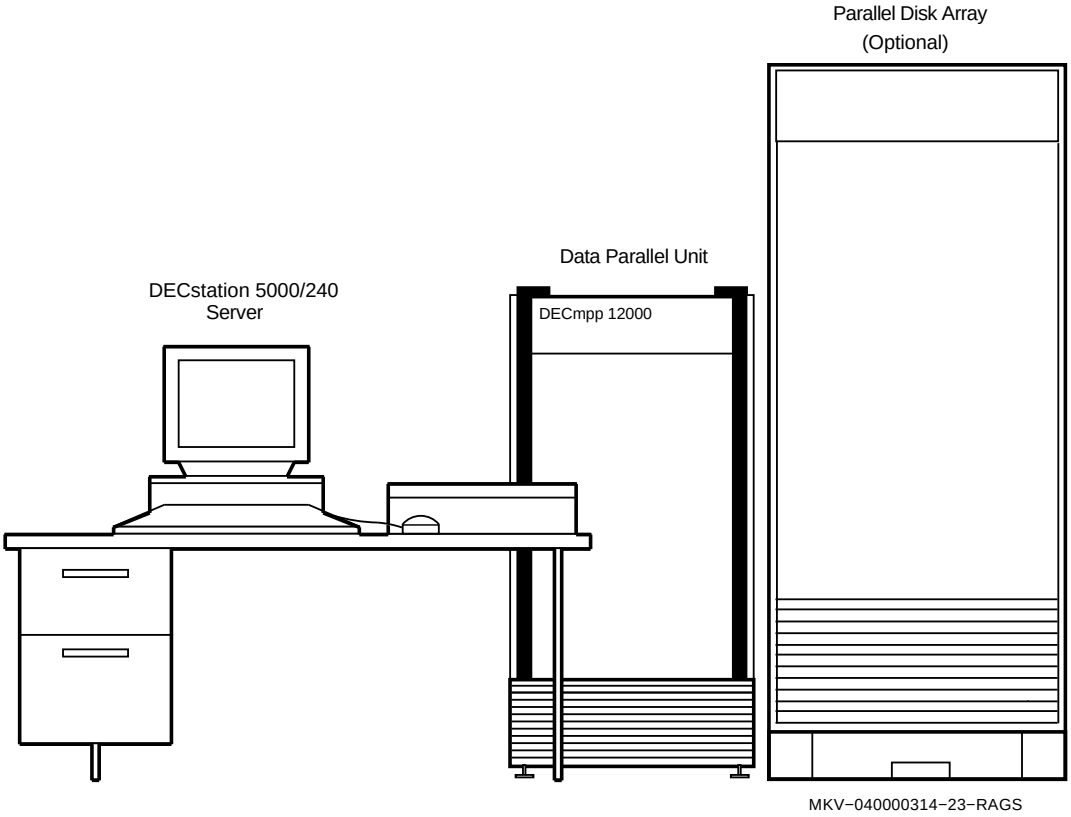


Figure 1–3 Typical DECmpp 12000–LC/Sx Installation



1.1 Turning the System On and Off

The DPU has three switches that control power: the keyswitch, the power selector, and the circuit breaker. Figure 2–1, Figure 2–2, and Figure 2–4 show their locations. Chapter 2 provides details on these switches.

The DPU keyswitch inside the front door has three positions: OFF, ON, and DIAGNOSTIC:

- OFF turns DPU power off.
- ON turns DPU power on.
- DIAGNOSTIC is similar to ON, but it also enables the VMEbus RESET button.

The DPU power selector switch has three positions: REMOTE, LOCAL, and OVERRIDE:

- REMOTE (lower position): Not used.
- LOCAL (middle position): Turns on the DPU independently of the server.
- OVERRIDE (straight up): Turns on the DPU, regardless of other conditions.

Warning

Do not use the OVERRIDE setting. It overrides critical safety systems. The OVERRIDE setting is for factory use only.

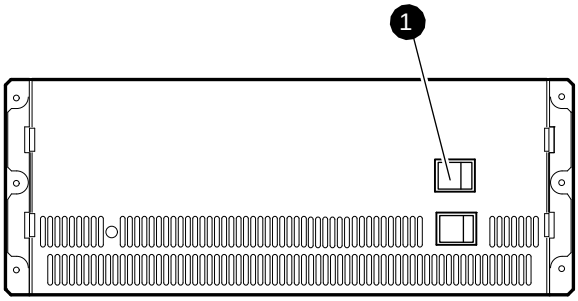
Table 1–1 shows the relationships between the DPU keyswitch and the DPU power selector.

Table 1–1 DPU Power Settings

DPU Power Selector	DPU Keyswitch	DPU Power
REMOTE	N/A	N/A
LOCAL	OFF	OFF
LOCAL	ON/DIAGNOSTIC	ON
OVERRIDE	Any setting	ON (Factory Use Only)

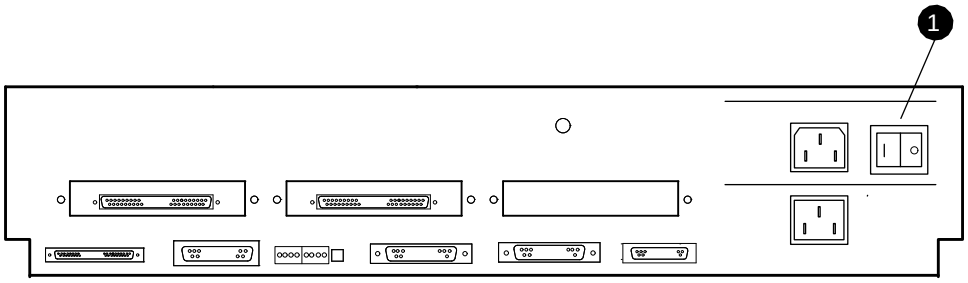
The power switch for the DECsystem 5900 server is on the front of the CPU drawer. It is labeled ❶ in Figure 1–4. The power switch for the DECstation 5000 server is at the rear of the CPU box. It is labeled ❶ in Figure 1–5. Both are rocker switches, labeled O and |. The server is ON when the | is pushed in.

Figure 1–4 DECsystem 5900 Power Switch



MKV-040000314-36-DG

Figure 1–5 DECstation 5000 Power Switch



MKV-040000314-50-DG

1.1 Turning the System On and Off

1.1.1 Powerup Sequence

Caution

To avoid unpredictable system operation, always turn the system components on or off in the correct sequence.

Take these steps to power up the DECmpp 12000/Sx:

1. Set the DPU rear circuit breaker to ON.
2. Turn on the DPU, setting the keyswitch to ON.
3. Turn on the Parallel Disk Array (PDA), if present in the configuration, by setting the rear circuit breaker to ON.
4. Turn on the server.
5. Boot the server.

When the system boots, it is ready to operate.

If you boot the server before you turn on the DPU, you cannot access the DPU. Any time you reconnect or cycle DPU power down and up, you must reboot the server.

1.1.2 Powerdown Sequence

Take these steps to power down the DECmpp 12000/Sx:

1. Halt the server, using either the `/etc/halt` or `/etc/shutdown` command.
2. Turn off the PDA, if present in the configuration, by setting the rear circuit breaker to OFF.
3. Turn off the DPU, setting the keyswitch to OFF.
4. Set the DPU rear circuit breaker to OFF.
5. Turn off the server.

Warning

Always set the circuit breaker at the rear of the DPU to OFF and unplug the power cord when working on the power supply or power system.

2

DPU Controls and Indicators

Chapter 2 provides descriptions and functional definitions of the data parallel unit (DPU) controls. Indicators on both the outside and the inside of the DPU show the status of the system and critical internal components.

Figure 2–1 (DECmpp 12000) and Figure 2–2 (DECmpp 12000–LC) show the locations of the DPU front controls and indicators. To access the front controls in either configuration, open the front door. The controls are located inside the enclosure at the top. Figure 2–3 provides a detail of the front controls.

Figure 2–4 shows the location of the rear controls and indicators. The DPU controls and indicators are described in detail in the following sections.

Figure 2–1 DECmpp 12000/Sx DPU Front Controls and Indicators

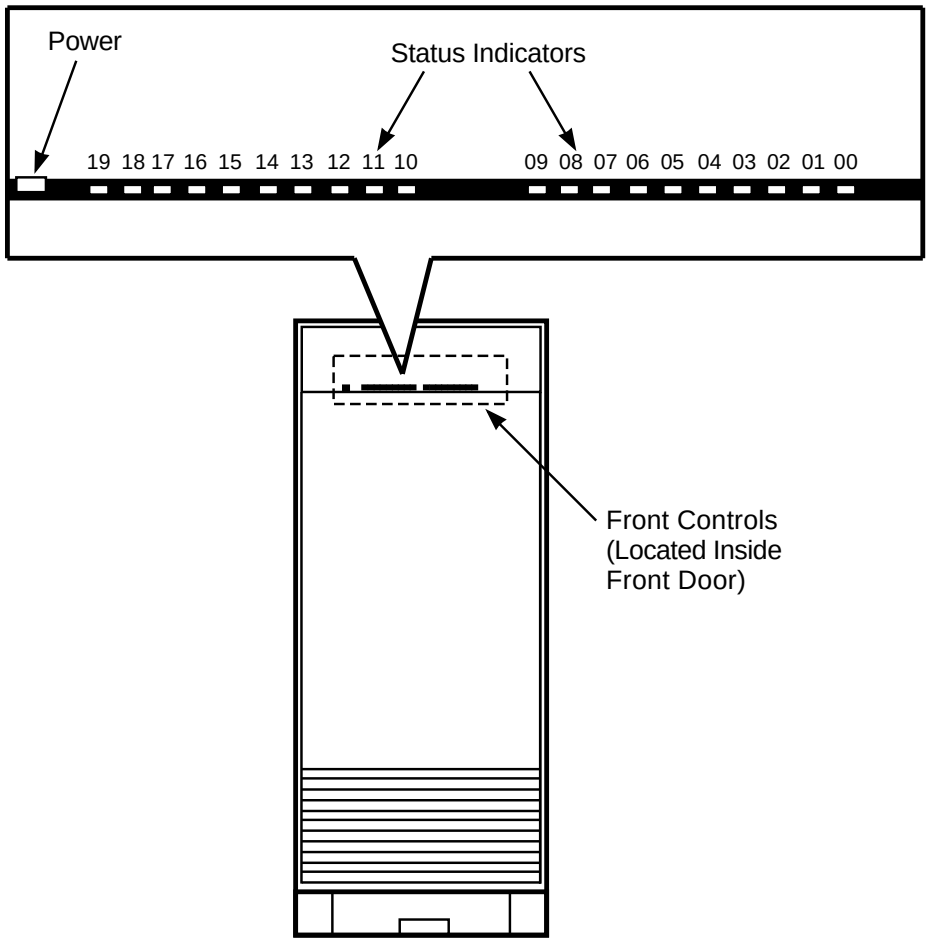


Figure 2–2 DECmpp 12000-LC/Sx DPU Front Controls and Indicators

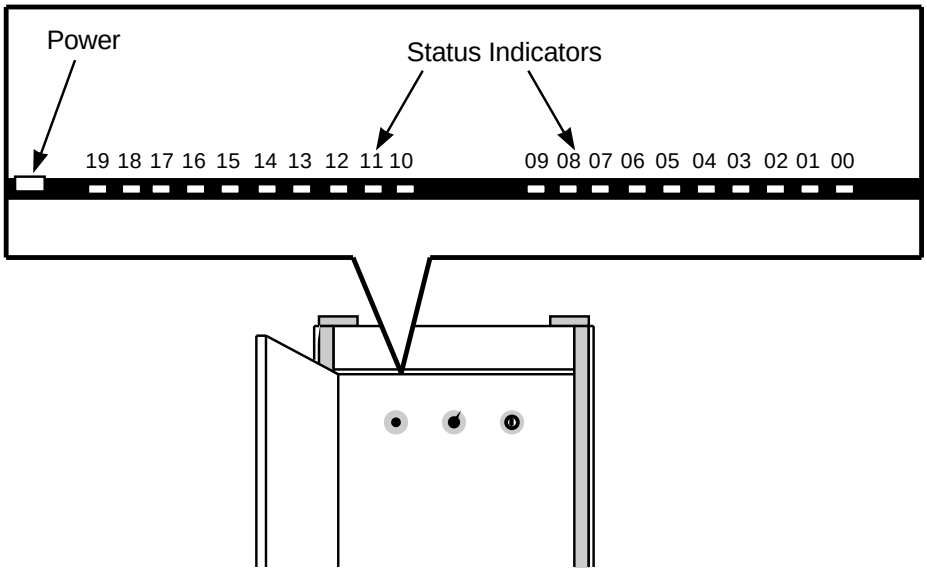


Figure 2–3 DPU Front Controls

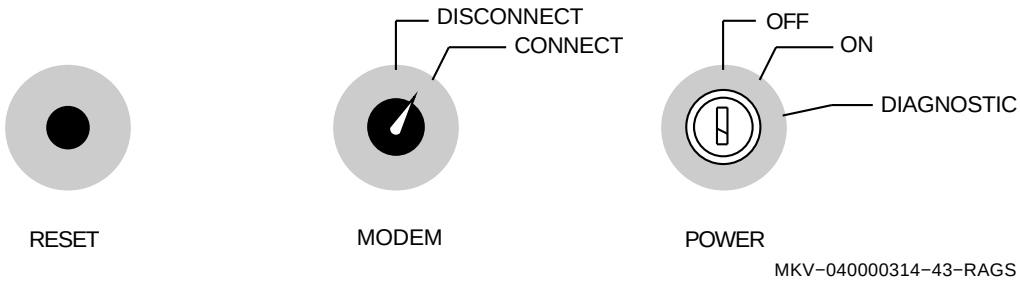
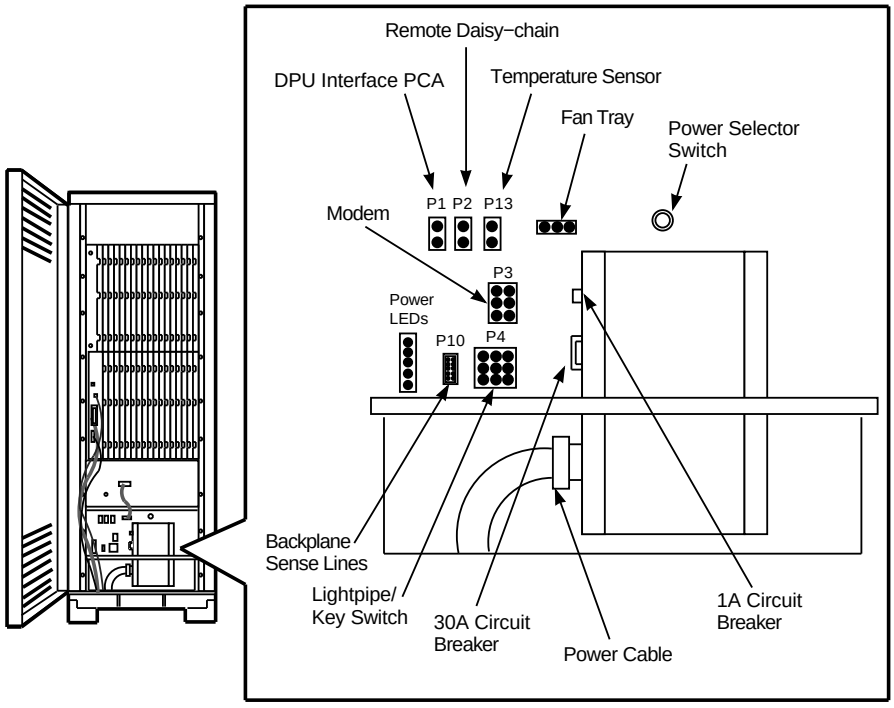


Figure 2–4 DPU Rear Controls and Indicators



2.1 Overview of Controls and Indicators

Several of the DECmpp 12000/Sx controls and indicators are described briefly in the list below. The rest are described in more detail in the following sections.

- POWER (keyswitch) — Controls power to the DPU and enables the VMEbus RESET button.
- VMEbus RESET pushbutton — Pushing this button resets the VMEbus only if the POWER keyswitch is set at DIAGNOSTIC.
- MODEM switch — At CONNECT, it enables the internal system modem (USA only). At DISCONNECT, it disables the modem.
- POWER indicator — Provides information about the power supplies and fan assembly, part of the front indicators.
- Power Selector — A three-position toggle switch, part of the rear controls.
- Service Indicators — Service indicators are described in detail in Section 2.3.
  - ACU Indicators — Provides information about the ACU PCB.
  - MVIB Indicator — Provides information about the Front-end VME interface (T6000) PCB.
  - PVME Indicators — Provides information about the PVME PCB.
  - PE and Router PCB Indicators — Provides information about the PE array and router.

2.2 DPU Power System

The DPU power system is complex, and you should understand it thoroughly before changing any settings.

Caution

To avoid system problems, do not turn the DPU off before bringing the front-end server to console or single-user mode.

The power system includes indicators that show the status of the various power supplies and a power sequencer that ensures the power supplies start in the correct order.

The DPU power system includes the following controls:

- The POWER keyswitch, shown in Figure 2–3, has three settings:
  - OFF (straight up) turns DPU power off.
  - ON (middle position) turns DPU power on when the power selector switch is set to LOCAL.
  - DIAGNOSTIC (lower position) is similar to ON, but also enables the VMEbus RESET button.

- The POWER SELECTOR switch is on the power tray panel inside the DPU enclosure at the rear. It is a toggle switch that must be pulled out slightly before it can be moved. It has three positions.
  - LOCAL (middle position): Allows the DPU to power up independent of the server.
  - OVERRIDE (straight up): Allows the DPU to power up regardless of other conditions.
  - REMOTE (lower position): Not used.
- Two circuit breakers are on the outside rear of the DPU power tray and are shown in Figure 2–4.
  - The rocker switch next to the power cable is a circuit breaker that is ON when it is UP. The circuit breaker is rated at 30 A for the DECmpp 12000 and 15 A for the DECmpp 12000–LC. The amber lamp above the rocker switch indicates there is power on both sides of the circuit breaker when it is lit.
  - The small rocker-type breaker controls the current to the circuits controlling the power sequencer. The amber lamp above the pushbutton indicates that there is power on both sides of the breaker. The breaker trips when it detects loads of more than 1 A. Push it in to reset it.
- The power status indicator on the DPU left front panel indicates the condition of the power supplies.
  - Yellow during power supply ramp-up
  - Green when all DPU DC voltages are correct

  When you first turn on the DPU, this indicator is yellow for approximately 3 seconds, then turns green. If it remains yellow, there is a power malfunction.
- A heat sensor shuts down all power supplies if it detects excess heat. The power status indicator changes to yellow. After cooling, the power supplies do not start again until the keyswitch is switched to OFF and then back to ON or DIAGNOSTIC. An indicator on the power tray also indicates excess temperature.
- When the DPU is turned on, the power sequencer turns on the -5 V power supply and verifies its correct operation before turning on the +5 V power supply. During this period (approximately 3 seconds), the power status indicator is yellow. When the sequencer turns on the +5 V power supply, the power status indicator turns green.

  If the power status indicator remains yellow, the powerup sequencer did not complete its sequence successfully. If this happens, switch the keyswitch to OFF and then back to ON or DIAGNOSTIC.

2.3 Indicators

The DPU has indicators on the front panel, the ACU PCB, the MVIB Front-end VME interface PCB (T6000), the PVME PVB, the PE array PCBs, the router PCBs, and on the power tray. All of these indicators provide information about the status of the system.

2.3.1 Front Panel Indicators

The DPU has 10 active indicators on the front panel, providing information about the state of the DPU (Figure 2–1 or Figure 2–2). Most of these are two-color indicators; they are either green, yellow, or off.

The front panel indicators are arranged in two banks of 10 indicators each. Table 2–1 lists the front panel indicators, starting from the left.

Table 2–1 Front Panel Indicators

Label	Green	Yellow	Off
Power	Power, fan, and temperature are OK	System powering up [1]	No AC power present
19	System run light: instructions are executing	Macrocode is not running	Not used
18	ACU is waiting for TOBEQ	Not used	ACU is not waiting for TOBEQ
17	ACU is waiting for FRBEQ	Page fault (overrides green)	ACU is not waiting for FRBEQ
16	PMem is using PE	PMem is not using PE	
15	Router is active	Not used	No router activity
14	I/O is taking place between PEs and I/O devices	Machine is temporarily stalled due to register interlock	One or more PEs are selected.
13	VME $\overline{AS}$ is active	Not used	VME $\overline{AS}$ is not active
12	VME $\overline{DTACK}$ is active	Not used	VME $\overline{DTACK}$ is not active
11	Not used	VME parity error	VME parity OK
10	Not used	Normally always on	Not used
9-0[2] [3]	Not used	Not used	Not used

[1]During normal system powerup, the power status indicator is yellow for a few seconds, then changes to green. If it remains yellow, the power sequencer did not complete powerup, and the power tray indicators indicate the reason for failure. When the power sequencer stops trying to power up the DPU, this indicator turns red.

[2]Indicators 0-4 cycle when the software daemon is running.

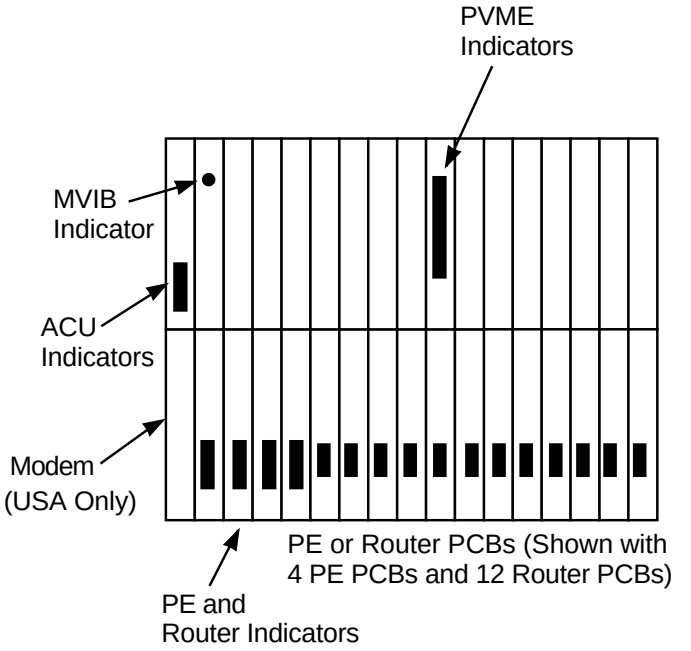
[3]Indicator 5 may become yellow if the background diagnostic tests fail.

2.3.2 Service Indicators

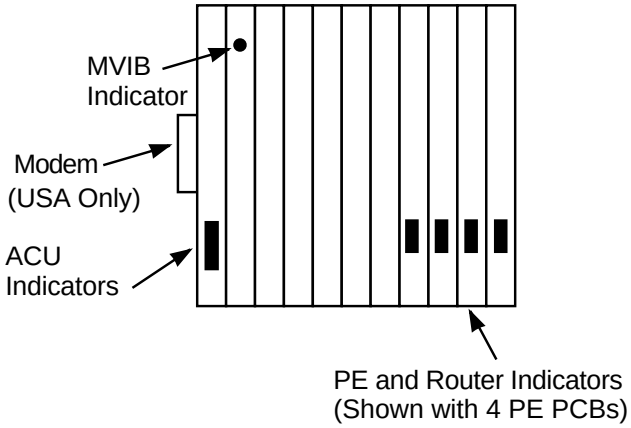
Service indicators include those on the various PCBs and on the power tray. Figure 2–4 shows the location of the power tray service indicators. Figure 2–5 shows the location of the PCB service indicators.

2.3 Indicators

Figure 2–5 PCB Service Indicator Locations



DECmpp 12000 Card Cage



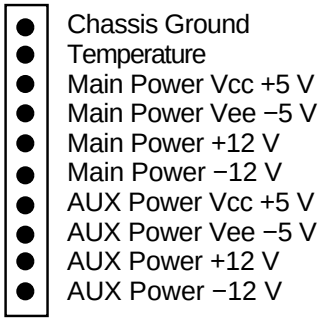
DECmpp 12000–LC Card Cage

MKV–040000314–54–RAGS

2.3.2.1 Power Tray Indicators

Figure 2–6 shows the ten indicators located on the DPU power tray rear panel. These indicators provide information about the power tray. They all show the status of some part of the power tray and are green during normal operation. If a failure is detected, the corresponding indicator changes to red, and the power tray shuts down. Any red indicators are latched ON, indicating the problem area, and initiate the power shutdown.

Figure 2–6 Power Tray Service Indicators



MKV−040000314−55−RAGS

Table 2–2 lists the function of each power tray indicator (from the top).

Table 2–2 Power Tray Service Indicators

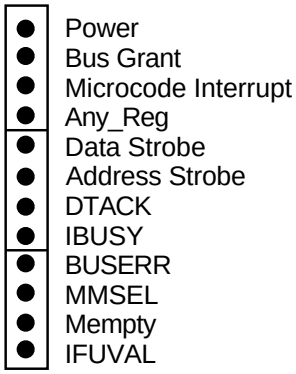
PS Indicator	Function
Chassis Ground	Green — Normal Red — Logic ground to chassis ground short; excess voltage detection
Temperature	Green — Normal Red — Excess temperature
Main Power Vcc +5 V	Green — Normal Red — Failure in +5 V supply
Main Power Vee –5 V	ON — Normal
Main Power +12 V	ON — Normal
Main Power –12 V	ON — Normal
Auxiliary Power Vcc +5 V	ON — 15 A or 30 A breaker is set ON OFF — indicates problem with power sequencer
Auxiliary Power Vee –5 V	ON — 15 A or 30 A breaker is set ON OFF — indicates problem with power sequencer
Auxiliary Power +12 V	ON — 15 A or 30 A breaker is set ON OFF — indicates problem with power sequencer
Auxiliary Power –12 V	ON — 15 A or 30 A breaker is set ON OFF — indicates problem with power sequencer

The chassis ground circuit detects differences in potential between chassis ground and logic ground and shuts down the power tray when the potential exceeds a preset 70 mV threshold.

2.3.2.2 Array Control Unit PCB Indicators

Figure 2–5 shows indicator locations on the array control unit (ACU) PCB. The 12 indicators on the ACU PCB are arranged in three groups of four (Figure 2–7). Table 2–3 lists the ACU indicator functions (from the top).

Figure 2–7 Array Control Unit PCB Indicators



MKV–040000314–56–RAGS

Table 2–3 ACU Indicators

Indicator	Function
Power	ON when the ACU PCB is powered
Bus Grant	ON when the ACU has VMEbus grant
Microcode Interrupt	ON when a microcode interrupt is in progress
Any_Reg	ON during a valid ACU access over the VMEbus
DSVME	Data strobe
ASVME	Address strobe
DTACK	ACU is generating DTACK
IBUSY	ACU is master of the bus
BUSERR	When ON, a VMEbus transaction did not complete or completed with an error. (An addressed VMEbus device did not respond within the VMEbus timeout limit (approximately 60 μsec) or returned an error signal in response to a VMEbus access.)
MMSEL	ON when the ACU is issuing current microcode from the M machine. (The current operation is a memory access, not a PE calculation.)
Mempty	ON when there are no pending memory operations in the M machine
IFUVAL	ON when the instruction fetch unit (IFU) is directed to a valid address. Reasons for an invalid IFU access are: – Attempting to execute code while a refresh cycle is in progress – A page fault – Attempting to execute code while the VMEbus is talking to instruction memory

2.3.2.3 PE Array and Router PCB Indicators

Each PE array PCB has eight indicators arranged into two 4-LED groups. The upper group provides status information about the PCB, and the lower bank displays the results of the global OR (GOR) for that PCB. Router PCBs have only the status indicators. Figure 2–5 shows the location of the PE and router indicators in the DPU. Figure 2–8 shows the indicator pattern. Table 2–4 lists the PE array and Router indicator functions (from the top).

Figure 2–8 PE and Router PCB Indicators

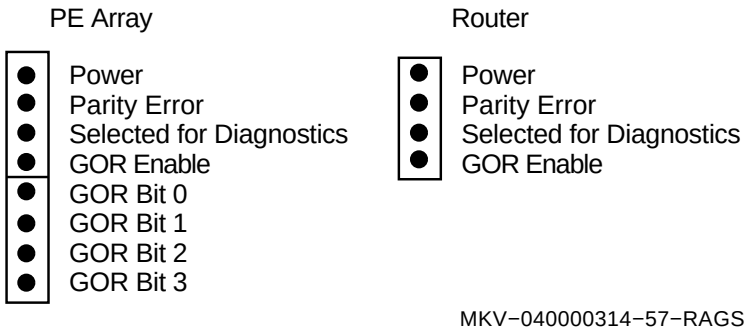


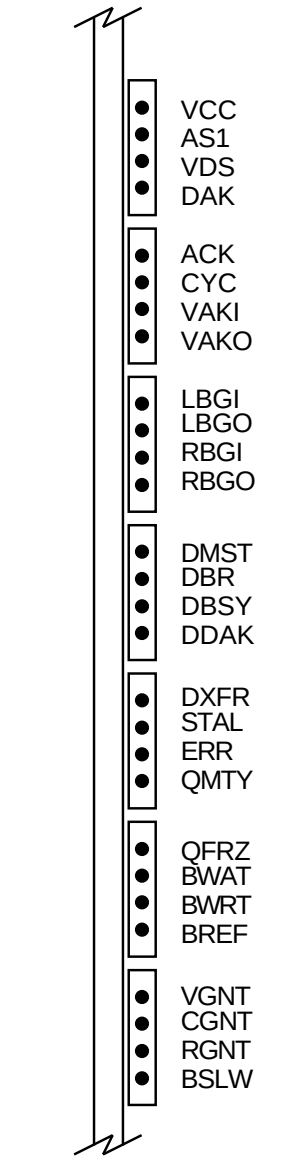
Table 2–4 PE Array and Router Indicators

Indicator	Function
Power	ON to indicate that the PCB is powered
Parity Error	ON when a parity error occurs, and OFF when the error-handling routine is completed or cleared by a reset
Selected for Diagnostics	ON when the PCB is selected by a diagnostic routine
GOR Enable	ON when the PCB GOR utility is selected
GOR Bit 0	ON when GOR bit 0 is active
GOR Bit 1	ON when GOR bit 1 is active
GOR Bit 2	ON when GOR bit 2 is active
GOR Bit 3	ON when GOR bit 3 is active

2.3.2.4 PVME Indicators

The PVME PCB has 28 signal indicators, as shown in Figure 2–9. Table 2–5 lists the signal name and the signal abbreviation used on the PCB. When the indicator is ON (green), the signal is true.

Figure 2–9 PVME Signal Indicators



MKV–040000314–02–RAGS

2.3 Indicators

Table 2–5 PVME Indicator Descriptions

Signal	Abbreviation	Description
Vcc	VCC	ON: +5 V power supply is ON
AS1_	AS1	ON: VMEbus AS (address strobe) is true
vmeDS	VDS	ON: VMEbus DS (data strobe) is true
DTACK_	DAK	ON: IOCTLR generated DTACK is true
bIACK	ACK	ON: VMEbus IACK (interrupt acknowledge) is true
IACKCYC	CYC	ON: IOCTLR is responding to an IACK VMEbus cycle
vmeIACKIN_	VAKI	ON: VMEbus interrupt acknowledge daisy chain-in is true
vmeIACKOUT_	VAKO	ON: VMEbus interrupt acknowledge daisy chain-out is true
vmeLBGIN_	LBGI	ON: Local VMEbus grant daisy chain-in is true
dmaLBBGOUT_	LBGO	ON: Local VMEbus grant daisy chain-out is true
vmeRBGIN_	RBGI	ON: Remote VMEbus grant daisy chain-in is true
dmaRBGOUT_	RBGO	ON: Remote VMEbus grant daisy chain-out is true
dmaMASTER	DMST	ON: IOCTLR is dma bus master
dmaBR_	DBR	ON: IOCTLR is asserting request for VMEbus
dmaBBSY_	DBSY	ON: IOCTLR is asserting VMEbus Busy
vmeDTACK_	DDAK	ON: VMEbus DTACK is true
chDXFR_	DXFR	N/A — May be ON or OFF (value is X)[1]
chSTALL_	STAL	N/A — May be ON or OFF (value is X)
chERR	ERR	N/A — May be ON or OFF (value is X)
queueEMPTY_	QMTY	N/A — May be ON or OFF (value is X)
queueFROZ	QFRZ	N/A — May be ON or OFF (value is X)
babWAIT_	BWAT	ON: IORAM is holding off access; address has crossed page boundary
babWRT_	BWRT	ON: IORAM is performing a write cycle
refGNT_	BREF	ON: IORAM is performing a refresh cycle
vmeGRANT_	VGNT	ON: VME Interface has been granted IORAM access
chGRANT_	CGNT	N/A — May be ON or OFF (value is X)
rioGRANT_	RGNT	ON: RIO Interface has been granted IORAM access
babSLOWDEV_	BSLW	ON: Slow VME device has accessed IORAM

[1] N/A = not applicable to PVME

3

Checking and Adjusting DPU Power Supply

Chapter 3 describes how to measure and adjust power supply voltage levels.

The data parallel unit (DPU) has four power supply output levels: +5 V, –5.2 V, +12 V, and –12 V. Measure these voltages at the DPU backplane, at the test points shown in Figure 3–1. For DECmpp 12000 systems adjust the voltage levels at the points shown in Figure 3–2. For DECmpp 12000–LC systems adjust the voltage levels at the points shown in either Figure 3–3 or Figure 3–4, depending on the power supply used in the system.

Warning

Physical tolerances are very tight at the power supply. 5 V current levels are in the 600 A range.

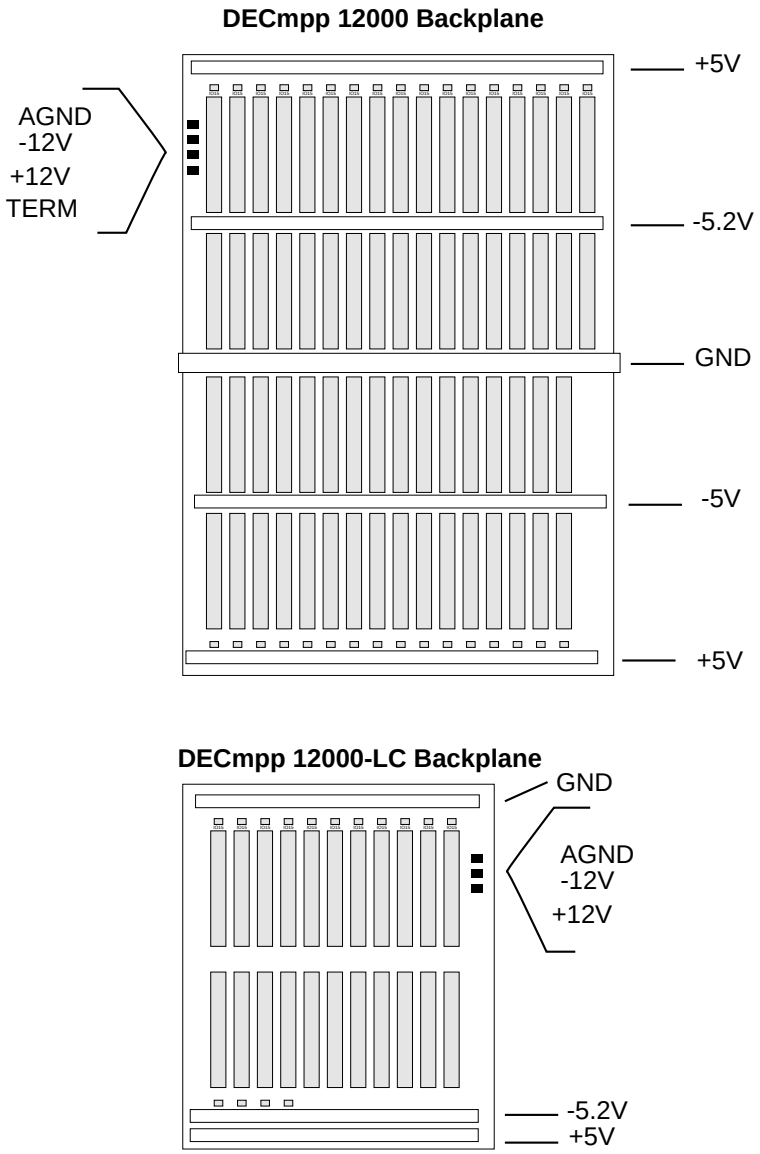
Do not short the power supply leads to ground or to other power supply leads.

Do not use conductive adjustment tools. Use only insulated tools.

When working around the power supply, do not wear loose clothing or jewelry, especially watches or rings.

Failure to observe these precautions can cause personal injury and damage to the equipment.

Figure 3–1 DPU Backplane Voltage Test Points



3.1 Voltage Measurements and Adjustments

For all voltage measurements and adjustments:

- Use a digital voltmeter to measure the supplies.
- Measure the voltages at the points shown in Figure 3–1.
- Use an insulated tool to make any adjustment.

Note

Adjust voltages only if they measure outside the tolerances specified in the following lists.

Follow these guidelines for adjusting the DPU power supplies:

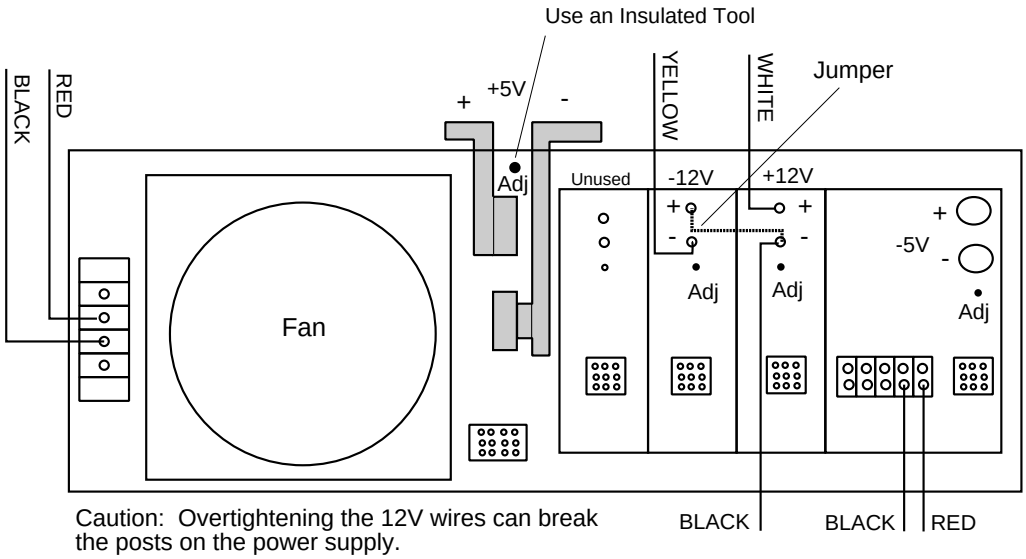
- +5 V supply
  - Adjust only if it is outside the range of +4.95 V to +5.05 V.
  - Adjust to exactly 5.0 V.
- –5.2 V supply
  - Adjust only if it is outside the range of –5.25 V to –5.15 V.
  - Adjust to exactly –5.2 V.
- –12 V supply
  - Adjust only if it is outside the range of –12.15 V to –11.85 V.
  - Adjust to exactly –12 V.
- +12 V supply
  - Adjust only if it is outside the range of +11.85 V to +12.15 V.
  - Adjust to exactly +12 V.

Figure 3–2 shows the power supply adjustment points and wiring for the DECmpp 12000 DPU.

Figure 3–3 and Figure 3–4 show the supply adjustment points and wiring for each of the power supplies used in the DECmpp 12000–LC DPU.

3.1 Voltage Measurements and Adjustments

Figure 3–2 DECmpp 12000 Power Supply Wiring and Voltage Adjustments



MKV-040000314-58-MPS

Figure 3–3 DECmpp 12000–LC HC Power Supply and Voltage Adjustments

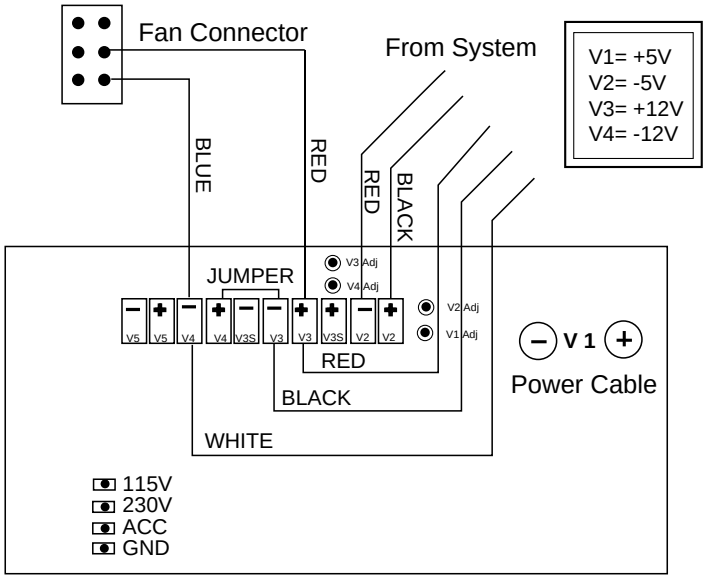
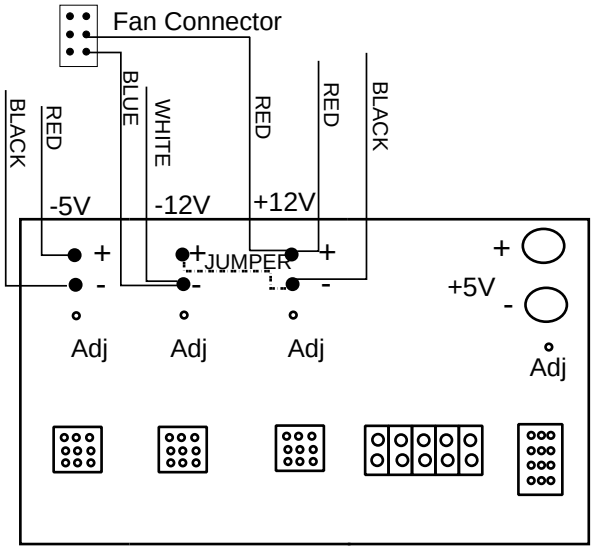


Figure 3–4 DECmpp 12000–LC Powertec Power Supply and Voltage Adjustments



Caution: Overtightening the 12V wires can break the posts on the power supply.

4

Cables, Connectors, and Auxiliary PCBs

Chapter 4 describes and identifies the connectors and cables in the DECmpp 12000/Sx system. It also describes PCBs that are not mounted in the data parallel unit (DPU) card cages.

4.1 Cables and Connectors

DECmpp 12000 systems require a 30 A, 250 V twist lock connector on a dedicated circuit in the United States. Systems shipped outside the United States require a 30 A, 220 Vac circuit and may be fitted with local cables and/or connectors during installation. The *DECmpp 12000/Sx Hardware Installation Guide* lists all optional power cable part numbers.

DECmpp 12000–LC systems must connect to either a dedicated 15 A, 110 Vac circuit or to a dedicated 10 A, 220 Vac or 240 Vac circuit, depending on local power availability.

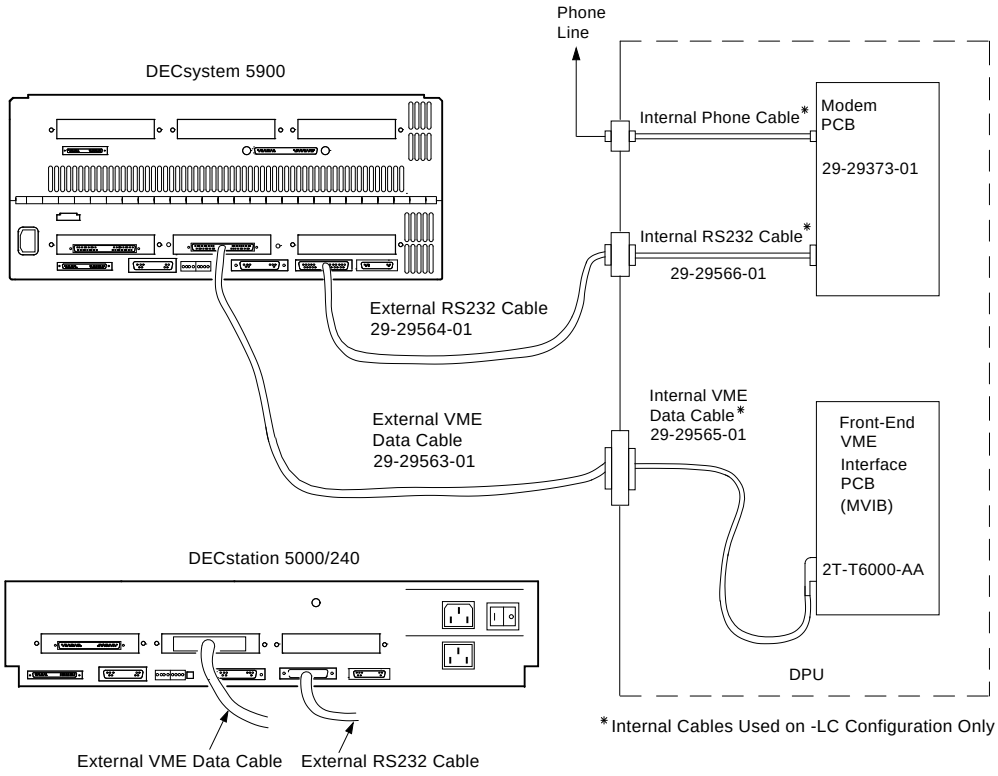
Following are brief descriptions of cables and connectors for the front-end server and the DPU. Refer to Figure 4–1 for a simplified diagram of the DECmpp system internal and external cable connections. The internal cables are used on the DECmpp 12000–LC configuration only.

- Telephone Connection — Internal DPU 2-foot cable to the modem; external cable length determined by installation site requirements (United States only).
- Data Cables

  Internal 4 foot cable with 100-pin connectors between the Front-end VME interface PCB (T6000) in the DPU card cage and the DPU rear panel interconnect PCB.

  External 15 foot cable with 100-pin connectors between the DPU rear panel cable connection and the server rear panel connection.
- Internal RS-232 Cable — Between the modem and the DPU rear panel interconnect PCB.
- External RS-232 Cable — Between the server rear panel and the DPU rear panel cable connection.
- Power Cable — In DECmpp 12000 systems, the power cable is wired directly to a terminal block inside the DPU power supply. DECmpp 12000–LC systems use a standard power cable.

Figure 4–1 DPU-Server Cables



4.2 Auxiliary PCBs

Auxiliary PCBs are defined as those not located in the DPU card cage. The DECmpp 12000/Sx Series DPUs have three auxiliary PCBs mounted inside the DPU enclosure.

- DPU Interconnect PCB (DECmpp 12000-LC/Sx configuration only)

  Provides a bulkhead connection for the VME and RS-232 cables. It is located inside the DPU, at the lower rear, behind the power supply.

- Lightpipe PCB

  Drives the LEDs on the DPU front indicator panel. In DECmpp 12000 systems this is mounted to the inside of the front door, along with the power switch assembly. In DECmpp 12000–LC systems it is mounted inside the DPU top cover, over the inner front door.

- Power Supply Controller PCB

  Controls the power sequencing system. This PCB is located inside the power tray enclosure.

5

Using Diagnostic Software

The DECmpp 12000/Sx contains diagnostics and related utility programs to test the array control unit (ACU) PCB, the processor element (PE) array PCBs, the PE array/router connections, the router PCB, EEPROMs, and memory. Appendix B provides reference pages for many of the diagnostics.

Note

Before running any DECmpp 12000/Sx diagnostics, make sure the front-end system is running correctly. Make sure no other users are on the system.

Running diagnostic programs may cause the system to crash, especially if the programs encounter problems.

To abort a diagnostic program while it is running, do not kill the process. Suspend it by entering Ctrl/C; this allows the diagnostic to restore the system to a usable state.

5.1 Diagnostic Environment

The diagnostic programs run under the ULTRIX operating system, version 4.2, and reside in the directory `$MP_PATH/field/bin`.

5.1.1 Running the Diagnostics

To run a diagnostic test, enter the name of the test followed by a space and any desired option, and then press Return. The default is a verbose mode that displays all related messages.

The options are as follows:

- `-C` — Customer environment [default] `[-q] [-t]`
- `-F` — Field environment; runs diagnostics in customer services mode `[-m] [-q] [-t]`
- `-M` — Manufacturing environment `[-l] [-m] [-q] [-s] [-t]`
- `-l` — Loop on error (Ctrl/C will break the loop)
- `-m` — Menu mode (provides a menu of subtests and parameter options)
- `-q` — Quick mode (some lengthy tests, such as `pe_arith` and `acu_clim`, are shortened)
- `-s` — Stop on error (User has the option to continue or abort the test)
- `-t` — Terse mode (displays only summary messages)

5.1.2 Suspending a Diagnostic Program

To suspend a diagnostic test, enter Ctrl/C. This suspends the current test and displays a menu similar to the following example:

```
TEST SUSPENDED AT USER'S REQUEST

1. Exit to the shell
2. Continue with the test

Select one:
```

Enter 1 or 2 to make the selection. The selections have the following effects:

- 1 — Exit to the shell

  The program returns from the diagnostic environment and leaves the system in good order for the next user.

- 2 — Continue with the test

  The diagnostic continues exactly what it was doing when Ctrl/C was pressed.

5.2 Types of Tests

DECmpp 12000/Sx diagnostics consist of two sets of programs: test suites and individual tests.

5.2.1 Test Suites

The DECmpp 12000/Sx contains two test suites, acu_diag and pe_diag. Run acu_diag, then pe_diag. The individual tests are described in Section 5.2.2. Total time for executing the full suite of diagnostics in a 4K PE system is approximately 45 minutes in default mode.

5.2.1.1 The acu_diag Test Suite

The acu_diag suite tests the ACU PCB. It runs the following tests in the order shown:

- acu_reg1
- met wcs
- acu_micr
- met cmem
- acu_reg2
- met imem
- met aux
- met map
- met pgtbl
- acu_macr
- acu_bound
- acu_pptest
- acu_clim

5.2.1.2 The pe_diag Test Suite

The pe_diag suite tests processor array PCBs and the backplane. It runs the following tests in the order shown:

- pe_scan
- pe_macro
- pe_ckxnet
- pe_ckonet
- pe_rtbp

5.2.2 Test Descriptions

This section briefly describes the diagnostic and utility programs:

- `acu_bound` — ACU Boundary Test

  Tests the ability of macro code to operate across page and row boundaries.

- `acu_clim`

  Tests the action of the ACU when certain CMEM limits and alignments are violated.

- `acu_diag`

  This test suite checks the ACU PCB. The tests in the suite are listed in Section 5.2.1.1.

- `acu_int`

  Tests the operation of interrupts.

- `acu_macro`

  Tests the ability of macro code to execute basic instructions.

- `acu_micro`

  Tests data paths, registers, and components on the ACU PCB.

- `acu_pgtbl`

  Tests the page table translation and comparison mechanism.

- `acu_pptest`

  Verifies the 32 general-purpose registers are functional; uses peeks and pokes.

- `acu_prof`

  Tests the three profile counters and the profile configuration register.

- `acu_reg1`

  Tests the ACU front-end registers.

- `acu_reg2`

  Tests the ACU front-end firmware-emulated registers.

- `acu_sup`

  Tests the hardware necessary to support user/supervisor mode firmware, CMEM address map, and software interrupt generation.

- `met`

  Tests memory. To run `met`, enter `met`, a space, the type of memory you wish to test, and press Return. If you need to run more detailed tests with `met`, you can get instructions by entering `met`, and then pressing Return.

  Memory Types are:

  - `umem` — Microcode memory (`wc` and `umem` are the same memory, accessed in different formats)
  - `cmem` — CMem
  - `imem` — IMem
  - `idma` — IMem DMA
  - `aux` — Auxiliary IMem
  - `map` — Map RAM
  - `pgtbl` — Page table
  - `lru` — Least-recently-used page reference memory
  - `preg` — PReg
  - `pmem` — PMem (1-Mbyte DRAM chips)
  - `qmem` — PMem (4-Mbyte DRAM chips)
  - `rio` — Router I/O; IORAM accessed via router system
  - `rpm` — Router parallel memory; high-speed RIO/PE test (`rp` uses the PE array to expand the test data by the number of PEs)
  - `vme` — VME memory; tests any memory on VME bus, especially auxiliary RAM PCB
  - `ioram` — IORAM; accessed via slave read/write over VME bus
  - `iodma` — IORAM DMA; IORAM is master, FE is slave
  - `iodmb` — IORAM DMA; IORAM is master, auxiliary RAM PCB is slave (`iodmb` is like `iodma`, but data is bounced off the VME RAM PCB)

- `pe_arith`

  Tests PE arithmetic functions.

- `pe_ckonet`

  Shifts data around successively larger octagonal loops among the PEs.

- `pe_ckxnet`

  Shifts data around the X-Net and checks for X-Net parity errors.

- `pe_diag`

  This test suite checks the PE array PCBs and the backplane. The tests in the suite are listed in Section 5.2.1.2.

- `pe_func`

  Tests various PE functions.

- `pe_macro`

  Tests the instruction set.

- `pe_memdiag`

  Tests PMem and reports the exact location of any errors found.

- `pe_rtbp`

  Tests various signal paths passing between PE array, backplane, and router PCBs. When it detects a failing path, it gives a complete report:

  - Signal name — as shown on schematic
  - Source PCB — chip pin number, backplane connector pin number
  - Destination PCB — chip pin number; backplane connector pin number

  Note

  The default mode is for a DECmpp 12000 system. If you run this test on a DECmpp 12000–LC system, enter 1100 after the test name.

- `pe_rtdiag`

  Tests the transmitting and receiving router functions.

- `pe_rtr`

  PE router wire test.

- `pe_scan`

  Tests the serial scan chains on the ACU, PE array, and router PCBs.

- `pe_xnet`

  Exhaustive test of all three X-Net instructions. If a failure occurs, it identifies the fault's PCB, chip, pin number, and backplane pin number.

- `rts`

  Tests the transmitting, receiving, and intermediate router functions. To run `rts`, `rtscfg` must be in the same directory.

- `rts13`

  Tests the transmitting and receiving router functions.

- `rts2`

  Tests the intermediate router functions.

5.3 Interpreting Log File Messages

Three logs provide useful information about the status of the DPU:

- `./LOG`
- `/usr/adm/dpujobmgr.log`
- `/etc/uerf`

Many of these logs have references to `fltcod`, which is the Fault Code Word. Its value provides a clue to the cause of an error, as described in Table 5–1.

Table 5–1 FLTCOD Values

Value	Error	Meaning
0x1	CLTSTOPARITY	Control store parity error
0x2	CTLPEPARITY	Control store PE parity error
0x4	CTLIOPARITY	Control store I/O parity error
0x10	CMEMLIMIT	CMem limit violation
0x20	PREGLIMIT	PReg limit violation
0x40	PMEMLIMIT	PMem limit violation
0x100	IMEMPARITY	Instruction memory parity error
0x200	RTRPARITY	Router parity error
0x400	XNETPARITY	X-Net parity error
0x800	RTRTNOR	Router T no R
0x1000	PREGALIGN	PReg alignment
0x2000	IMEMALIGN	IMem alignment
0x4000	CMEMALIGN	CMem alignment
0x10000	ILLOP	Illegal operation
0x80000	BUSERR	VMEbus error
0x100000	MACHFAULT	Machine fault
0x200000	IRR	I/O instruction error
0x400000	RPROTO	Router protocol error
0x800000	ILLPEADDR	Illegal PE address
0x1000000	DIV0	Integer divide by zero
0x8000000	FINEXACT	Floating-point inexact result
0x10000000	FINVOP	Floating-point invalid operand
0x20000000	FDIV0	Floating-point divide by zero
0x40000000	FUNFLO	Floating-point underflow
0x80000000	FOVFLO	Floating-point overflow

5.3 Interpreting Log File Messages

5.3.1 The ./LOG File

All diagnostic programs scroll output to the screen and copy it to the LOG file in the current directory (directory in which you executed diagnostics). If a LOG file already exists, diagnostic output is appended to it.

If you do not have write-permission in the directory from which you are running the diagnostics, the LOG file is not changed.

Most tests produce more than a screenful of output, and you may want to edit the LOG. The LOG files can become quite large, especially in the default verbose mode. The `grep` utility is useful for displaying keywords and obtaining a quick summary. For example, entering `grep FAILED LOG` and pressing Return produces a list of all the failure messages.

LOG output and contents are similar to the following examples. LOG file messages are described below.

Example of Standard Messages:

```
Start of diagnostic test: ACU HARDWARE VME REGISTERS TEST (acu_reg1)
Timestamp: Mon May  6 09:43:43 1991

CUSTOMER ENVIRONMENT:
          Standard length tests
          Standard messages

$Revision: 1.5 $
PASSED.... ECSR register test: 0 errors
PASSED.... Page Table registers test: 0 errors
FAILED.... Instruction Memory register test: 1 error
PASSED.... Master Interrupt Registers Test: 0 errors
FAILED.... Instruction DMA Registers test: 2 errors
PASSED.... Microcode Registers Test: 0 errors
PASSED.... Serial Scan Registers test: 0 errors
PASSED.... MAPCSR Register Test: 0 errors
SUMMARY... ACU HARDWARE VME REGISTERS TEST (acu_reg1). 3 errors, 0 aborts

End of diagnostic test: ACU HARDWARE VME REGISTERS TEST (acu_reg1)
Timestamp: Mon May  6 09:43:46 1991
```

Example of Terse Messages:

```
Start of diagnostic test: ACU HARDWARE VME REGISTERS TEST (acu_reg1)
Timestamp: Mon May  6 09:44:06 1991

CUSTOMER ENVIRONMENT:
          Standard length tests
          Terse messages

$Revision: 1.5 $
SUMMARY... ACU HARDWARE VME REGISTERS TEST (acu_reg1). 3 errors, 0 aborts

End of diagnostic test: ACU HARDWARE VME REGISTERS TEST (acu_reg1)
Timestamp: Mon May  6 09:44:09 1991
```

The following list contains standard LOG file messages and brief explanations:

- NOTE

  Provides information about the environment in which the test is running. It describes what the test is doing, so you understand the context if there is an error.

- CAUTION

  An abnormal result, but not an actual error. This could indicate trouble elsewhere; this is often the case when there are unused bits not at 0.

- ERROR

  Something is actually wrong. For example, the test might have written 1 to a bit and read back 0.

- FAILED

  The reporting test received one or more ERRORs.

- ABORTED

  The test terminated abnormally before completion.

- WARNING

  The program encountered a serious situation associated with an area not being directly tested. This situation does not necessarily cause the current test to fail, but it can provide a clue to other problems. This is a more serious message than CAUTION.

- PASSED

  The selected test completed successfully.

5.3.2 The /usr/adm/dpujobmgr.log File

This file is a log of all the dpumanager daemon activity. This log reports background diagnostic errors, register status when errors are reported, and ACU kernel information. This log is especially helpful when you are trying to determine what might have caused a problem when a program aborts.

This example shows there was a control store parity error (FLTCOD=2). Control store parity errors can occur on a PE array PCB, ACU PCB, or the backplane.

```
(dpu0) Tue Apr 30 07:04:11 1991 (21) DPU fault: SWOPT=5; HWOPT=5;
FLTCOD=0x2; PMSTAT=0x0; PMEMECC=0
```

For this type of error, look at the parity LED on the PE PCBs for a clue to which PCB is generating the error.

This example shows an execution of a `mpshutdown` command and the execution of a `dpumanager` command.

```
(dpu0) Mon May 13 10:39:44 1991 Termination signal received; shutting down
(dpu0) Tue Jul 21 15:40:10 1992 Starting up; Version 2.2.0
(dpu0) Tue Jul 21 15:40:11 1992 loading microcode file: "/usr/mpp/etc/mp12ucode.wo"
(dpu0) Tue Jul 21 15:40:18 1992 loading ACU kernel file: "/usr/mpp/etc/acuk"
(dpu0) Wed Jul 22 12:10:55 1992 ACU kernel timeout (command 6)
ECSR=0x4002, QCSR=0x0, PTACCESS=0, CPC=0xffff00a8
(dpu0) Wed Jul 22 12:10:55 1992 Save of context failed, killing job; pid = 10333
(dpu0) Wed Jul 22 12:10:55 1992 ACU kernel command error (!ECSR<Run>)
(dpu0) Wed Jul 22 12:10:55 1992 unable to abort user -- reloading ACU kernel
(dpu0) Wed Jul 22 12:10:55 1992 Job context lost in system reset; pid = 10333
(dpu0) Wed Jul 22 12:10:55 1992 Job context lost in system reset; pid = 10355
(dpu0) Wed Jul 22 12:10:55 1992 loading ACU kernel file: "/usr/mpp/etc/acuk"
(dpu0) Wed Jul 22 12:10:56 1992 (@1) DPU fault: SWOPT=4; HWOPT=4;
FLTCOD=0x200; PMSTAT=0x0; PMEMECC=0
(dpu0) Wed Jul 22 12:10:56 1992 (@2) 15 PEs had errors
(dpu0) Wed Jul 22 12:10:56 1992 (@2) PE fault: PE number=0x678 (board
5, cluster 3,6, PE-in-cluster 0,0); error bits=0x2 ( ROUTER )
(dpu0) Wed Jul 22 12:10:56 1992 (@2) PE fault: PE number=0x674 (board
5, cluster 3,5, PE-in-cluster 0,0); error bits=0x2 ( ROUTER )
(dpu0) Wed Jul 22 12:10:56 1992 (@2) PE fault: PE number=0x64c (board
4, cluster 3,3, PE-in-cluster 0,0); error bits=0x2 ( ROUTER )
(dpu0) Wed Jul 22 12:10:56 1992 (@2) PE fault: PE number=0x638 (board
1, cluster 3,6, PE-in-cluster 0,0); error bits=0x2 ( ROUTER )
(dpu0) Wed Jul 22 12:10:56 1992 (@2) PE fault: PE number=0x634 (board
1, cluster 3,5, PE-in-cluster 0,0); error bits=0x2 ( ROUTER )
(dpu0) Wed Jul 22 12:10:56 1992 (@2) PE fault: PE number=0x60c (board
0, cluster 3,3, PE-in-cluster 0,0); error bits=0x2 ( ROUTER )
(dpu0) Wed Jul 22 12:10:56 1992 (@2) PE fault: PE number=0x27c (board
5, cluster 1,7, PE-in-cluster 0,0); error bits=0x2 ( ROUTER )
(dpu0) Wed Jul 22 12:10:56 1992 (@2) PE fault: PE number=0x278 (board
5, cluster 1,6, PE-in-cluster 0,0); error bits=0x2 ( ROUTER )
(dpu0) Wed Jul 22 12:10:56 1992 (@2) PE fault: PE number=0x260 (board
5, cluster 1,0, PE-in-cluster 0,0); error bits=0x2 ( ROUTER )
(dpu0) Wed Jul 22 12:10:56 1992 (@2) PE fault: PE number=0x25c (board
4, cluster 1,7, PE-in-cluster 0,0); error bits=0x2 ( ROUTER )
(dpu0) Wed Jul 22 12:10:56 1992 (@2) PE fault: PE number=0x23c (board
1, cluster 1,7, PE-in-cluster 0,0); error bits=0x2 ( ROUTER )
(dpu0) Wed Jul 22 12:10:56 1992 (@2) PE fault: PE number=0x238 (board
1, cluster 1,6, PE-in-cluster 0,0); error bits=0x2 ( ROUTER )
(dpu0) Wed Jul 22 12:10:56 1992 (@2) PE fault: PE number=0x21c (board
0, cluster 1,7, PE-in-cluster 0,0); error bits=0x2 ( ROUTER )
(dpu0) Wed Jul 22 12:10:56 1992 (@2) PE fault: PE number=0x44 (board 4,
cluster 0,1, PE-in-cluster 0,0); error bits=0x2 ( ROUTER )
(dpu0) Wed Jul 22 12:10:56 1992 (@2) PE fault: PE number=0x4 (board 0,
cluster 0,1, PE-in-cluster 0,0); error bits=0x2 ( ROUTER )
(dpu0) Wed Jul 22 18:24:32 1992 (@1) DPU fault: SWOPT=3; HWOPT=4;
FLTCOD=0x400000; PMSTAT=0x0; PMEMECC=0
(dpu0) Wed Jul 22 18:24:32 1992 (@2) 1822 PEs had errors (only 21 reported)
(dpu0) Wed Jul 22 18:24:32 1992 (@2) PE fault: PE number=0x1c (board 0,
cluster 0,7, PE-in-cluster 0,0); error bits=0x2 ( ROUTER )
(dpu0) Wed Jul 22 18:24:32 1992 (@2) PE fault: PE number=0x1b (board 0,
cluster 0,6, PE-in-cluster 0,3); error bits=0x2 ( ROUTER )
```

```
(dpu0) Wed Jul 22 18:24:32 1992 (@2) PE fault: PE number=0x1a (board 0,
cluster 0,6, PE-in-cluster 0,2); error bits=0x2 ( ROUTER )
(dpu0) Wed Jul 22 18:24:32 1992 (@2) PE fault: PE number=0x19 (board 0,
cluster 0,6, PE-in-cluster 0,1); error bits=0x2 ( ROUTER )
(dpu0) Wed Jul 22 18:24:32 1992 (@2) PE fault: PE number=0x18 (board 0,
cluster 0,6, PE-in-cluster 0,0); error bits=0x2 ( ROUTER )
(dpu0) Wed Jul 22 18:24:32 1992 (@2) PE fault: PE number=0x14 (board 0,
cluster 0,5, PE-in-cluster 0,0); error bits=0xf ( XNET ROUTER PMEM/HARD PMEM/SOFT )
(dpu0) Wed Jul 22 18:24:32 1992 (@2) PE fault: PE number=0x13 (board 0,
cluster 0,4, PE-in-cluster 0,3); error bits=0x8 ( PMEM/SOFT )
(dpu0) Wed Jul 22 18:24:32 1992 (@2) PE fault: PE number=0x12 (board 0,
cluster 0,4, PE-in-cluster 0,2); error bits=0x7 ( XNET ROUTER PMEM/HARD )
(dpu0) Wed Jul 22 18:24:32 1992 (@2) PE fault: PE number=0x11 (board 0,
cluster 0,4, PE-in-cluster 0,1); error bits=0x7 ( XNET ROUTER PMEM/HARD )
(dpu0) Wed Jul 22 18:24:32 1992 (@2) PE fault: PE number=0x10 (board 0,
cluster 0,4, PE-in-cluster 0,0); error bits=0x5 ( XNET PMEM/HARD )
(dpu0) Wed Jul 22 18:24:32 1992 (@2) PE fault: PE number=0xf (board 0,
cluster 0,3, PE-in-cluster 0,3); error bits=0xd ( XNET PMEM/HARD PMEM/SOFT )
(dpu0) Wed Jul 22 18:24:32 1992 (@2) PE fault: PE number=0xe (board 0,
cluster 0,3, PE-in-cluster 0,2); error bits=0x7 ( XNET ROUTER PMEM/HARD )
(dpu0) Wed Jul 22 18:24:32 1992 (@2) PE fault: PE number=0xd (board 0,
cluster 0,3, PE-in-cluster 0,1); error bits=0x7 ( XNET ROUTER PMEM/HARD )
(dpu0) Wed Jul 22 18:24:32 1992 (@2) PE fault: PE number=0xc (board 0,
cluster 0,3, PE-in-cluster 0,0); error bits=0xe ( ROUTER PMEM/HARD PMEM/SOFT )
(dpu0) Wed Jul 22 18:24:32 1992 (@2) PE fault: PE number=0xa (board 0,
cluster 0,2, PE-in-cluster 0,2); error bits=0x6 ( ROUTER PMEM/HARD )
(dpu0) Wed Jul 22 18:24:32 1992 (@2) PE fault: PE number=0x9 (board 0,
cluster 0,2, PE-in-cluster 0,1); error bits=0x6 ( ROUTER PMEM/HARD )
(dpu0) Wed Jul 22 18:24:32 1992 (@2) PE fault: PE number=0x8 (board 0,
cluster 0,2, PE-in-cluster 0,0); error bits=0xf ( XNET ROUTER PMEM/HARD PMEM/SOFT )
(dpu0) Wed Jul 22 18:24:32 1992 (@2) PE fault: PE number=0x7 (board 0,
cluster 0,1, PE-in-cluster 0,3); error bits=0x2 ( ROUTER )
(dpu0) Wed Jul 22 18:24:32 1992 (@2) PE fault: PE number=0x6 (board 0,
cluster 0,1, PE-in-cluster 0,2); error bits=0xc ( PMEM/HARD PMEM/SOFT )
(dpu0) Wed Jul 22 18:24:32 1992 (@2) PE fault: PE number=0x5 (board 0,
cluster 0,1, PE-in-cluster 0,1); error bits=0xc ( PMEM/HARD PMEM/SOFT )
(dpu0) Wed Jul 22 18:24:32 1992 (@2) PE fault: PE number=0x4 (board 0,
cluster 0,1, PE-in-cluster 0,0); error bits=0x3 ( XNET ROUTER )
(dpu0) Wed Jul 29 10:54:21 1992 Starting up; Version 2.2.0
(dpu0) Wed Jul 29 10:54:22 1992 loading microcode file: "/usr/mpp/etc/mp12ucode.wo"
(dpu0) Wed Jul 29 10:54:41 1992 loading ACU kernel file: "/usr/mpp/etc/acuk"
(dpu0) Tue Aug 11 08:19:50 1992 Starting up; Version 2.2.0
(dpu0) Tue Aug 11 08:19:51 1992 loading microcode file: "/usr/mpp/etc/mp12ucode.wo"
(dpu0) Tue Aug 11 08:20:12 1992 loading ACU kernel file: "/usr/mpp/etc/acuk"
```

This example shows a hangup with the ACU kernel. The kernel timed out and automatically restarted.

```
(dpu0) Tue Apr 30 15:43:00 1991 ACU kernel timeout (command 15)
ECSR=0x4002, QCSR=0x0, PTACCESS=0, CPC=0xffff00a8
(dpu0) Tue Apr 30 15:43:00 1991 unable to run background diagnostics --
reloading ACU kernel
(dpu0) Tue Apr 30 15:43:00 1991 loading ACU kernel file:
"/usr/maspar/etc/acuk"
```

This example shows the log after a correctable PMEM error occurred. If PMEMECC=6 or more, a hard error is recorded, and the program aborts.

```
(dpu0) Mon May 13 05:37:49 1991 (2) 1 PE had errors
(dpu0) Mon May 13 05:37:49 1991 (2) PE fault: PE number=0xa5f (board 4,
cluster 5,7, PE-in-cluster 0,3); error bits=0x8 ( PMEM/SOFT )
(dpu0) Mon May 13 06:46:01 1991 (4) PMEM parity errors detected;
PMEMECC=1 (in rollpmem)
```

5.3.3 The /etc/uerf File

All errors are reported in this log. This is very helpful when the dpumanager is not running.

To look at the log, enter the following commands:

```
# cd /etc
# uerf -R | more
```

In the log examples below, Example 1 is a message from the DPU; FLTCOD 200 indicates a router parity error. Example 2 is an example of an ULTRIX error message; it has nothing to do with the DPU, but is shown here as a contrast to the DPU message.

Example 1:

```
----- EVENT INFORMATION -----

EVENT CLASS                             OPERATIONAL EVENT
OS EVENT TYPE                  250.     ASCII MSG
SEQUENCE NUMBER                 29.
OPERATING SYSTEM                        ULTRIX 32
OCCURRED/LOGGED ON                      Wed Jul 22 12:10:54 1992 EDT
OCCURRED ON SYSTEM                      mpdemo.mps.m
SYSTEM ID                x82020230      HW REV: x30
                                        FW REV: x2
                                        CPU TYPE: R2000A/R3000
PROCESSOR TYPE                          KN02/R3000
MESSAGE                                 pc=0xffff127c, fltcod=0x200
```

Example 2:

```
----- EVENT INFORMATION -----

EVENT CLASS                             OPERATIONAL EVENT
OS EVENT TYPE                  250.     ASCII MSG
SEQUENCE NUMBER                 37.
OPERATING SYSTEM                        ULTRIX 32
OCCURRED/LOGGED ON                      Thu Jul 23 01:03:06 1992 EDT
OCCURRED ON SYSTEM                      mpdemo.mps.m
SYSTEM ID                x82020230      HW REV: x30
                                        FW REV: x2
                                        CPU TYPE: R2000A/R3000
PROCESSOR TYPE                          KN02/R3000
MESSAGE                                 NFS server mpsg not responding, still
                                         _trying
```

6

Removal and Replacement Procedures

Chapter 6 describes how to remove and replace the following data parallel unit (DPU) components:

- Array control unit (ACU) Printed Circuit Board (PCB)
- Front-end VME interface PCB (T6000)
- Processor element (PE) array PCB
- Router PCB
- Power/status indicator PCB
- Power tray
- Fan tray

Appendix A provides a list of recommended spares for DECmpp 12000/Sx Series Data Parallel Units.

6.1 Card Cage Access

The ACU, PE array, front-end VME interface, and router PCBs are accessed from the rear of the DPU. Figure 6–1 shows the card cage access for a DECmpp 12000 DPU. Figure 6–2 shows the card cage access for a DECmpp 12000–LC DPU.

To open the DPU rear door, use the hex wrench supplied with the system.

6.1 Card Cage Access

Figure 6–1 DECmpp 12000/Sx Card Cage Access

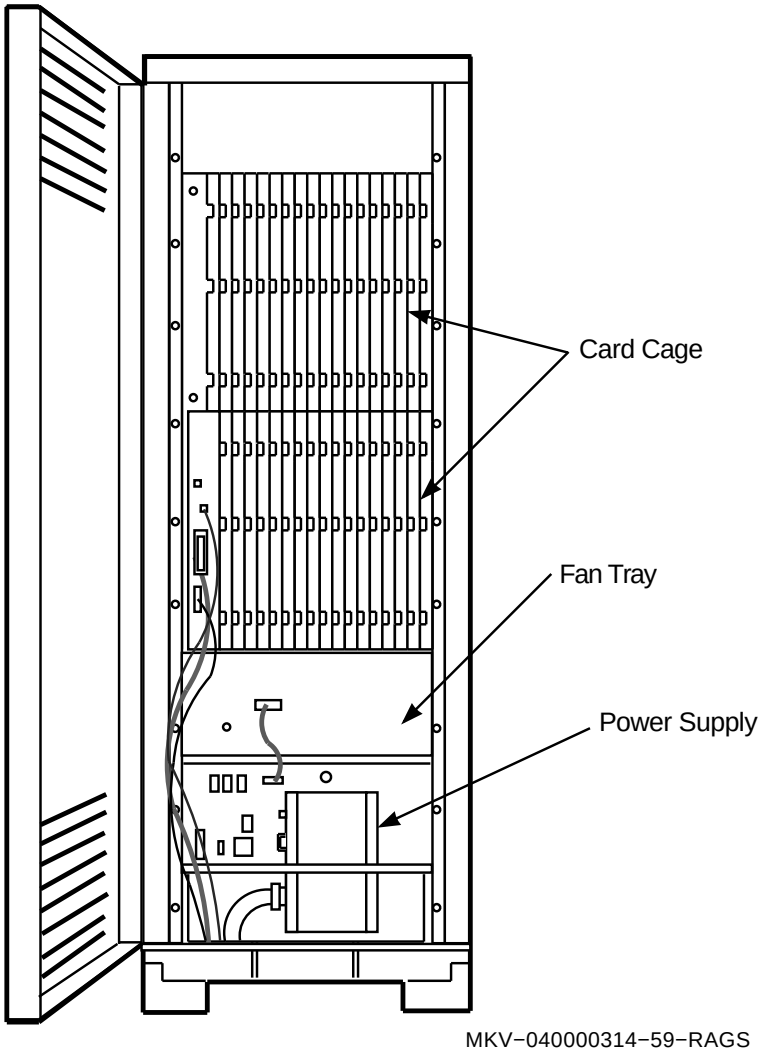
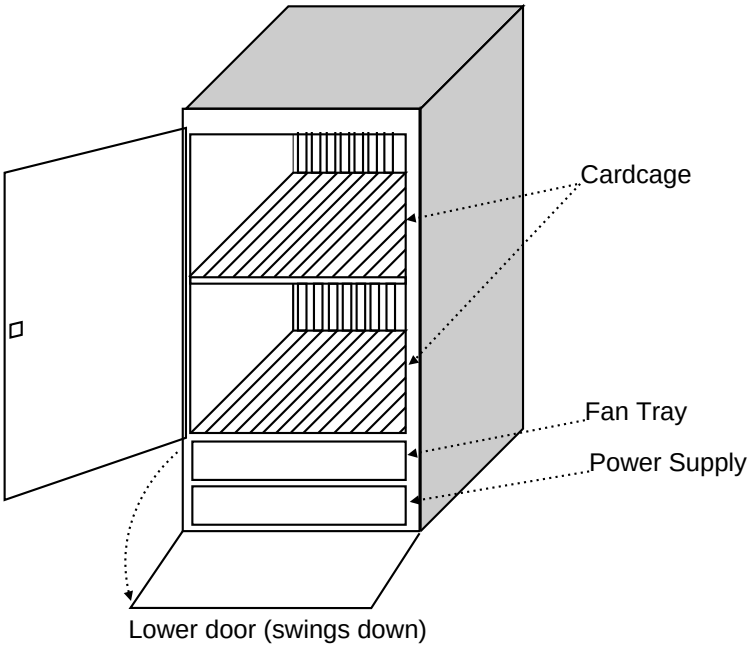


Figure 6–2 DECmpp 12000–LC/Sx Card Cage Access



6.2 DPU Card Cage Slots

The DPU card cage is divided into two dedicated blocks: one for I/O PCBs and the other for PE array PCBs, as shown in Figure 6–3. The DECmpp 12000 supports the ACU, the VME interface PCB, 16 PE array PCBs, and 15 I/O PCBs. The DECmpp 12000–LC supports the ACU, the VME interface PCB, 4 PE array PCBs, and 5 I/O PCBs.

6.2.1 I/O Slots

The first I/O slot, labeled ACU, always holds the ACU PCB. The second I/O slot (slot IO00) is configured to support the Front-end VME interface PCB (MVIB). This is a T6000 PCB mounted on a Parallel VME 6U Adapter PCB. The other slots are for the PVME PCB and VMEbus PCBs mounted on Parallel VME 6U Adapters.

In the DECmpp 12000 configuration, physical slot 10 (slot IO08) is reserved for the PVME controller PCB and physical slot 6 (slot IO04) is reserved for the PDA interface.

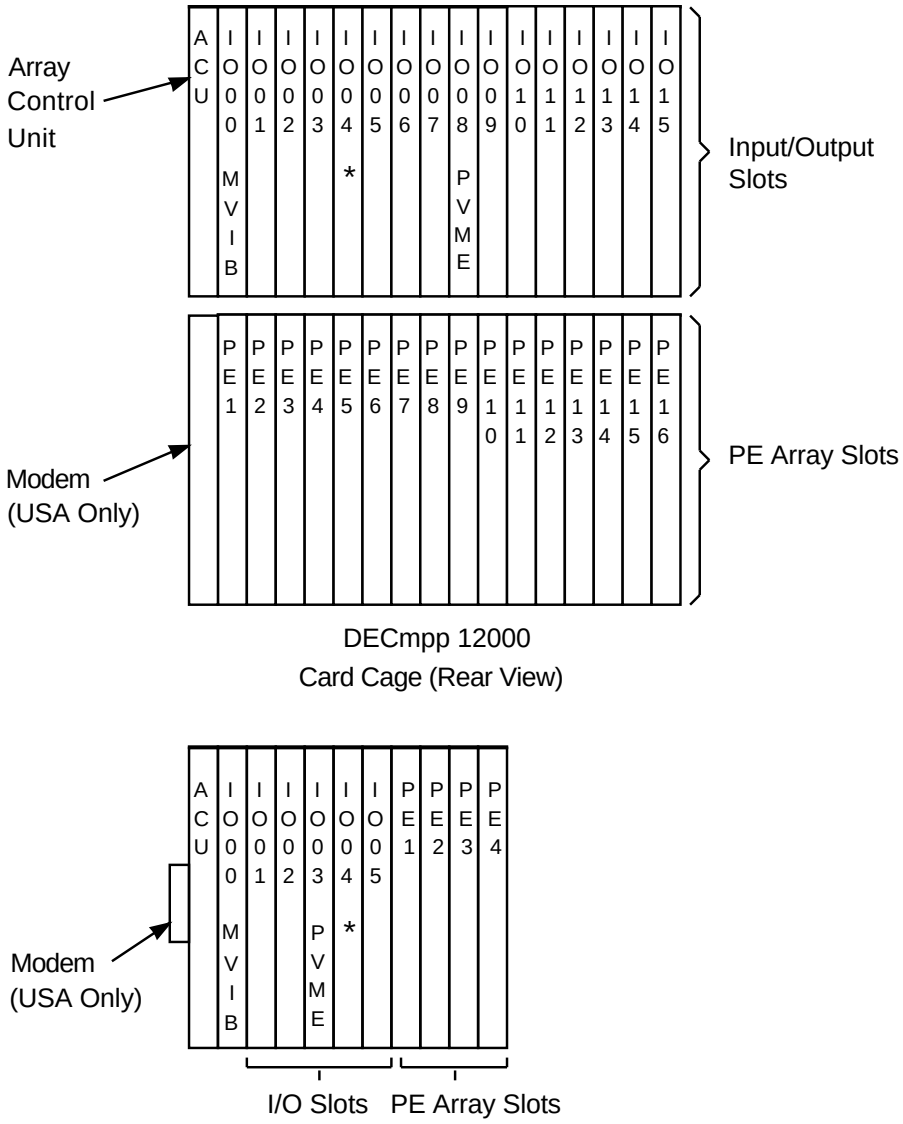
In the DECmpp 12000–LC configuration, physical slot 5 (slot I/O 03) is reserved for the PVME controller and physical slot 6 (slot IO04) is reserved for the PDA interface.

6.2.2 PE Array PCB Slots

DECmpp 12000 systems support 1, 2, 4, 8, or 16 PE array PCBs; DECmpp 12000–LC systems support 1, 2, or 4 PE array PCBs. The PE array PCBs must be in contiguous slots, starting with the first PE array PCB slot. If the number of PE array PCBs changes, backplane jumpers must be reconfigured, as described in Chapter 7.

All PE array PCB slots that do not contain a PE array PCB must contain a router PCB. The DECmpp will not work if any PE array PCB slot is empty.

Figure 6–3 DPU Card Cage Slots



* Slot IO04 Reserved for Optional PDA Interface

MKV−040000314−03−RAGS

6.3 Replacing DPU Card Cage PCBs

Warning

To avoid personal injury or damage to equipment, make sure the system is turned off before replacing or adding any PC PCBs. 5 V power supplies deliver current in the 600 A range.

Proper antistatic protection must be worn while servicing the DPU.

To avoid damage, always handle a PCB correctly:

- Wear antistatic equipment (for instance, a wrist or ankle strap) when handling a PCB.
- If working on a PCB, use a static-free work surface.
- Handle it gently by the edges.
- Never handle a PCB by the component or etched surfaces.
- Store PCBs in a static-proof container.

Each PCB in the DPU card cage has ejector levers on the top and bottom of the PCB faceplate, shown in Figure 6–4. Always use the ejector levers when removing or replacing PCBs:

- To release the PCB, lift up the lower end of the top lever, and press down on the upper end of the bottom lever, moving the PCB partway out of its slot.
- To secure the PCB in the card cage slot, press down the lower end of the top lever and the upper end of the bottom lever.

6.3.1 Replacing the Array Control Unit PCB

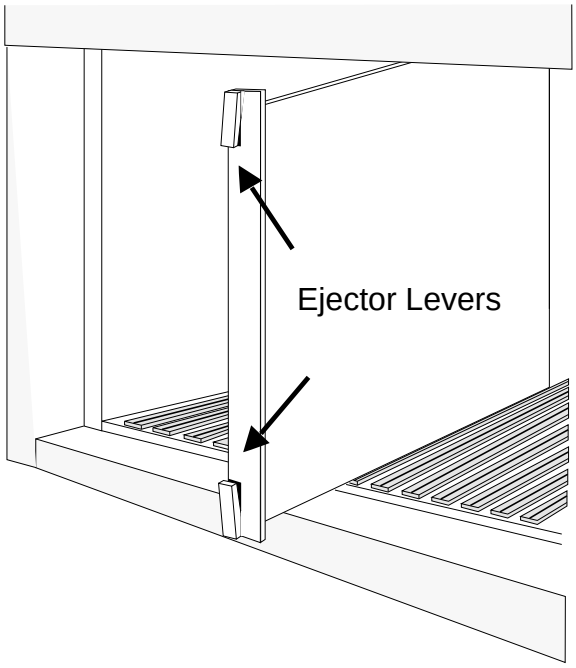
Caution

Make sure that you are grounded through a wrist or ankle strap before handling the ACU.

The ACU PCB is in the first I/O slot (labeled ACU) in the DPU card cage. To replace the ACU PCB, take the following the steps.

1. Turn off the system, as described in Chapter 1.
2. Open the DPU rear door.
3. Loosen the captive screws on the top and bottom of PCB faceplate.
4. Remove the top PCB retainer bar.
5. Use the ejector levers shown in Figure 6–4 to release the PCB and move it outward from its slot.
6. Carefully remove the PCB from its slot, handling it by the edges, and place it in a static-proof container.
7. Verify the jumper configuration on the new ACU PCB (Table 6–1).
8. Carefully slide the replacement PCB into the slot.

Figure 6–4 PCB Ejector Levers



MKV-040000314-37-MPS

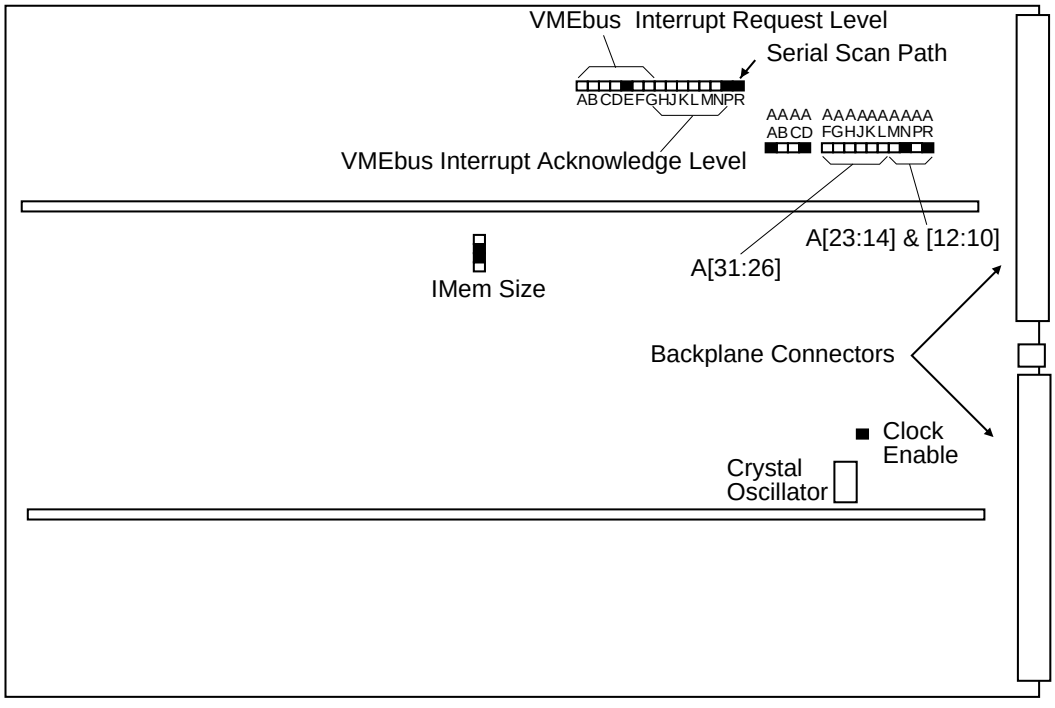
9. With the ejector levers open (out), seat the PCB, and push the levers closed. Although you may have to push firmly to seat the PCB, do not force it.
10. Secure the PCB in the card cage, tightening the captive screws on the top and bottom of faceplate.
11. Install the upper PCB retainer bar.
12. Close and latch the DPU rear door.
13. Turn on DPU power, as described in Chapter 1.

Figure 6–5 shows the location of the ACU jumpers. The darkened blocks indicate installed jumpers.

Table 6–1 ACU Jumper Settings

Jumper Function	Factory Setting
ACU VMEbus address	0xFCC000000
ACU VMEbus interrupt request level	Level 1
ACU VMEbus interrupt acknowledge level	Level 1
Independent scan path selection from the server (for diagnostics only)	Off
Selection of ACU as a VMEbus arbiter	ACU VMEbus arbiter selected
16-bit or 32-bit word selection for ACU instruction DMA	32-bit
Block or pipelined mode for ACU instruction DMA	Pipelined
IMem size	1 MB
ACU PCB clock enable/disable	Enabled

Figure 6–5 ACU Jumpers



Note: Darkened Jumpers
are INSTALLED.

MKV-040000314-38-MPS

6.3.2 Replacing Front-End VME Interface PCB

To replace the front-end VME interface PCB, follow these steps:

1. Turn off the system, as described in Chapter 1.
2. Open the DPU rear door.
3. Disconnect the 100-pin AMP connector on the PCB faceplate.
4. Remove the upper PCB retainer bar.
5. Loosen the captive screws at the top and bottom of the PCB faceplate.
6. Use the ejector levers shown in Figure 6–4 to release the PCB and move it outward from its slot.
7. Carefully remove the PCB from its slot, handling it by the edges, and place it in a static-proof container.
8. Carefully slide the replacement PCB into the slot.
9. With the ejector levers open (out), seat the PCB, and push the levers closed. Although you might have to push firmly to seat the PCB, do not force it.
10. Secure the PCB in the card cage, tightening the captive screws on the top and bottom of the faceplate.
11. Install the upper PCB retainer bar.
12. Reconnect the 100-pin AMP connector to the front of the PCB.
13. Close and latch the DPU rear door.
14. Turn on the power to the DPU, as described in Chapter 1.

6.3.3 Replacing PE Array and Router PCBs

To replace PE array or router PCBs, follow these steps:

1. Turn off the system, as described in Chapter 1.
2. Open the DPU rear door.
3. Loosen the captive screws at the top and bottom of the PCB faceplate.
4. Use the ejector levers shown in Figure 6–4 to release the PCB and move it outward from its slot.
5. Remove the lower PCB retainer bar.
6. Carefully remove the PCB from its slot, handling it by the edges, and place it in a static-proof container.
7. Carefully slide the replacement PCB into the slot.
8. With the ejector levers open (out), seat the PCB, and push the levers closed. Although you may have to push firmly to seat the PCB, do not force it.
9. Secure the PCB in the card cage, tightening the captive screws on the top and bottom of faceplate.
10. Install the lower PCB retainer bar.
11. Close and latch the DPU rear door.
12. From the front, open the DPU front door(s), and make sure that the DPU backplane jumpers are installed correctly, as described in Chapter 7.
13. Close the DPU front door(s).
14. Turn on the power to the DPU, as described in Chapter 1.

Note

When removing the PE PCBs and the router PCBs, make sure that the slot number that the PCB was removed from is noted on the defective module tag.

6.4 Replacing DPU Power Trays

This section contains instructions for removing and installing the DPU power trays.

6.4.1 Removing the DECmpp 12000/Sx Power Tray

Caution

It is important to label all cables as you disconnect them. Reinstalling cables with the wrong polarity can render the system inoperative.

Follow these steps to remove the power tray:

1. Shut down the server and turn off the power to the server and the DPU.
2. Open the back door of the DPU.
3. Set the DPU 30 A circuit breaker OFF, and unplug the power cord from the power source.
4. To avoid possible problems when powering up, use the ejector levers shown in Figure 6–4 to remove each PCB from the backplane, in both the upper and lower card cages.
5. Disconnect all the cables at the back of the power tray (Figure 6–6).
6. Open the DPU front door. Figure 6–7 shows a front view of the power tray.
7. Using two 9/16-inch open-end wrenches, remove the +5 V positive and negative power bars at the front of the power tray. Note the offset on the negative post for reinstallation. This offset is needed for clearance between the positive and negative posts.

   To avoid damage to the power post on the –12 V supply, do not use a deep socket when removing the positive +5 V bar.
8. Using a 9/16-inch wrench, remove the cables from the –5 V supply.
9. Using a 5/16-inch wrench, remove the +12 V and –12 V cables.
10. Cut any cable restraints fastening cables to the power tray.
11. Remove four screws in the front and two screws in the rear that secure the power tray to the enclosure frame and air plenum.
12. Make sure you have disconnected all cables from the back of the power tray. Do not let the power cables get caught on the power tray while sliding it out or in. Cuts in cable insulation can cause direct shorts.
13. From the DPU front, gently slide the power tray forward, checking that no cables are getting caught on the power tray. Make sure the attached AC power cord does not get caught coming through the rear skirt assembly. Also, make sure that the gasket on the fan tray above does not get caught and tear away.
14. Do not place any weight on the back of the power tray.

6.4 Replacing DPU Power Trays

Figure 6–6 DECmpp 12000 DPU Power Tray Rear

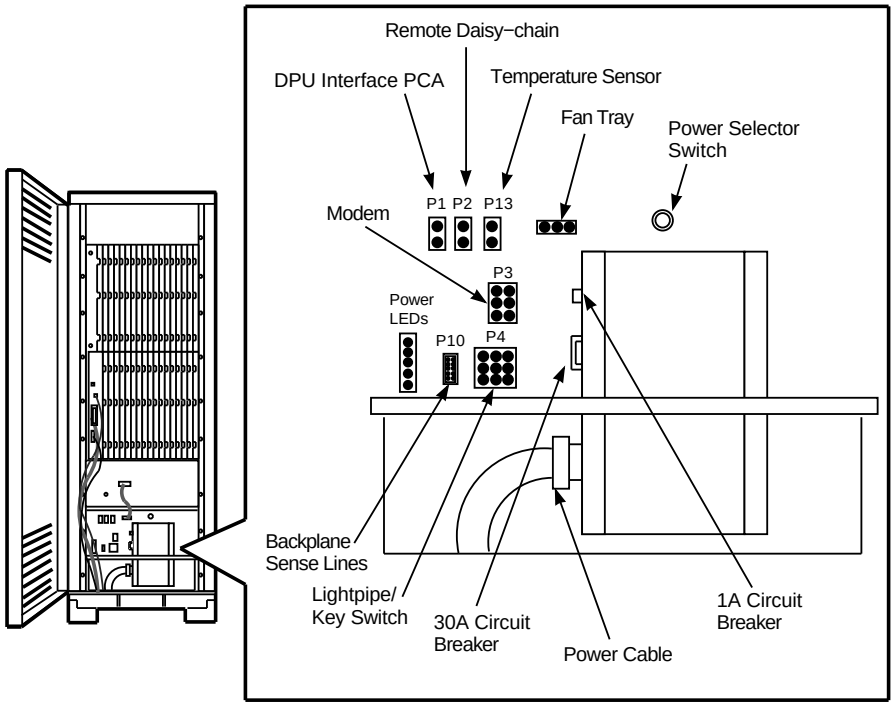
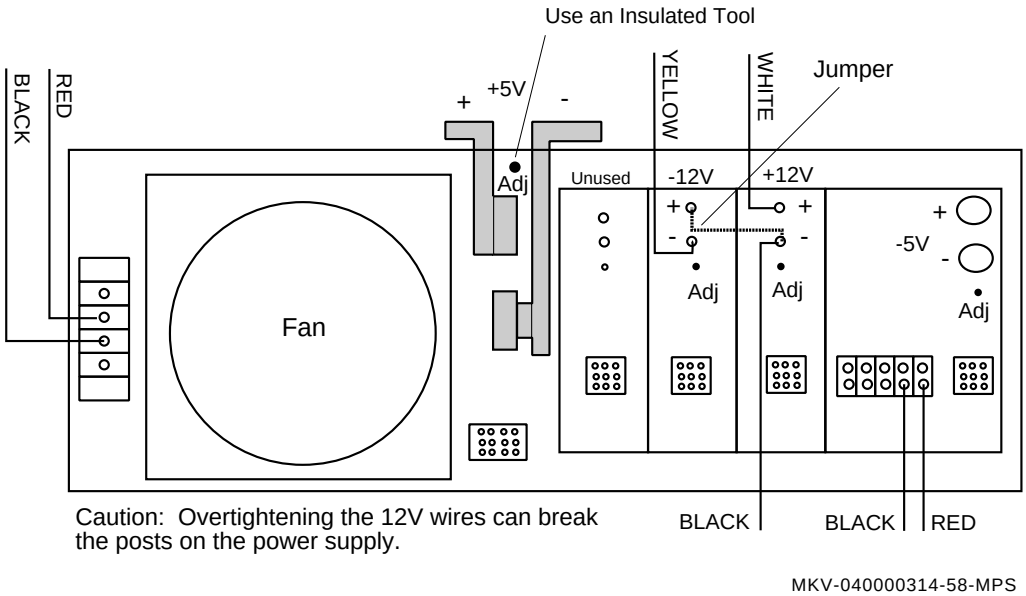


Figure 6–7 DECmpp 12000 DPU Power Tray Front



6.4.2 Installing the DECmpp 12000/Sx Power Tray

Follow these steps to install the power tray:

1. Gently slide the new power tray into place, feeding the AC power cord through the DPU rear skirt assembly. Again, make sure that the fan tray gasket does not get caught. Before sliding the tray in the last few inches, make sure that there are no cables in the back of the DPU blocking its way.
2. Replace six screws, securing the power tray to the frame and air plenum.
3. Connect the +12 V and –12 V cables to the power supply. To avoid snapping the power post, apply minimum force when tightening the nut.
4. Install the –5 V cables. The cable going to the top post (+) is actually ground if you didn't mark the cable earlier.
5. Install the +5 V power bars. It is easiest to install the positive bar first, followed by the negative bar.
6. Connect 2-pin connectors to the back of the power tray:
   a. Left connector from the DPU interface PCA
   b. Middle connector for daisy-chaining other machines
   c. Right connector from the temperature sensor
7. Make sure that all cables are connected properly and install cable restraints, fastening cables to the power tray.
8. Before proceeding, ensure that all PCBs have been removed from the backplane.
9. Ensure that there are no shorts in the DC voltage supply system. With an ohmmeter, check the +5 V, –5 V, +12 V, and –12 V connections with respect to ground. A zero ohm reading indicates a short and must be investigated before applying power.
10. Plug the power cable into the source of power.
11. Set the 30 A circuit breaker to ON.
12. Set the power selector switch set to LOCAL.
13. Turn on power to the DPU with the keyswitch.
14. With a digital voltmeter, measure all DC voltage levels. Measure the voltages on the backplane as described in Chapter 3. If needed, adjust the appropriate levels on the power supply. Make sure you have checked all voltage levels.
15. Turn off the DPU.
16. Reinstall and secure all card cage PCBs using the ejector levers, seating them firmly into the backplane.
17. Turn on the power to the DPU.
18. Remeasure the DC voltages and fine tune them if necessary.
19. Close the front door.
20. Turn on the server.
21. Run diagnostics and applications to verify the system operation.

6.4.3 Removing the DECmpp 12000–LC/Sx Power Tray

Caution

It is important to label all cables as you disconnect them. Reinstalling cables with the wrong polarity can render the system inoperative.

Follow these steps to remove the power tray:

1. Shut down the DECstation and turn off power to the workstation and the DECmpp 12000–LC, as described in Chapter 1.
2. Set the DPU breaker switch to OFF, and unplug the power cord from the power source.
3. Open the DPU back door.
4. To avoid possible problems when powering up, use the ejector levers to detach each PCB from the backplane.
5. Disconnect all cables at the back of the power tray.
6. Open the DPU front door and remove the front inner door.
7. Using a 9/16-inch open-end wrench or socket wrench, remove the cables from the +5 V power supply.
8. Remove the –5 V, +12 V, and –12 V wires from the power supplies:
   - Powertec — Using a 5/16-inch socket wrench, remove the –5 V, +12 V, and –12 V wires
   - HC Power — Using a Phillips screwdriver, remove the –5 V, +12 V, and –12 V wires
9. Remove the four Phillips Screws holding the power tray to the frame.
10. Make sure you have removed all of the cables from the back of the power tray.

Caution

Do not let the power cables get caught on the power tray when sliding it in or out. If the cable insulation is cut, it can cause a direct short.

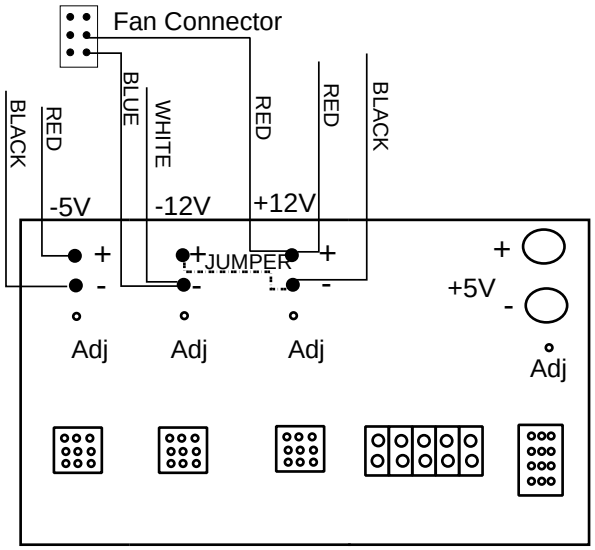
11. From the DPU front, carefully lift the power tray cables out of the way while pulling the power tray out of the enclosure.
12. Lay the power tray down flat or on its side. Do not put weight on the back of the tray.

6.4.4 Installing the DECmpp 12000–LC/Sx Power Tray

Take the following steps to install the power tray:

1. Slide the new power tray into place.
2. Lift the power supply cables out of the way and slide in the power tray.
3. Replace the four Phillips Screws, fastening the power tray to the frame.
4. Replace the –5 V, +12 V, and –12 V wires:
   - Powertec power supply — Use a 5/16-inch socket wrench to secure the nuts after the wires are in place; to avoid snapping the power post, apply minimum force when tightening the nuts. Refer to Table 6–2 and Figure 6–8 when connecting the wires.
   - HC Power supply — Use a Phillips screwdriver to secure the wires to the power supply, referring to Table 6–3 and Figure 6–9.
5. Connect the +5 V cables.
6. Connect the cables on the back of the power tray.
7. Before turning on the system power, make sure that all the cables have been installed properly and ensure that there are no shorts in the DC voltage supply system. With an ohmmeter, check the +5 V, –5 V, +12 V, and –12 V connections with respect to ground. A zero ohm reading indicates a short and must be investigated before applying power.
8. Turn on the power to the DPU.
9. Measure the voltage levels, and, if needed, fine tune the levels on the power supplies. Measure the voltages at the backplane. Voltage settings are described in Chapter 3.
10. Turn the DPU power off.
11. Using the ejector levers, replace the card cage PCBs, seating them firmly into the backplane.
12. Turn the DPU power on.
13. On the backplane, measure the power supply voltages again, and adjust if necessary.
14. Close the DPU doors.
15. Run diagnostics and application programs to verify proper performance.

Figure 6–8 Powertec Power Supply



Caution: Overtightening the 12V wires can break the posts on the power supply.

Table 6–2 Powertec Power Supply Wiring

Voltage	Value	Wire	Connection
–5 V	+	Black	Ground from backplane
	–	Red	– 5 V to backplane
–12 V	+	Black	Jumper to +12 V (–) terminal
	–	White	–12 V to backplane
	–	Blue	Ground for fan tray
+12 V	+	Red	+12 V to backplane
	+	Red	To fan tray
	–	Black	Ground to backplane for ±12 V
	–	Black	Jumper to –12 V (+) terminal

Figure 6–9 HC Power Supply

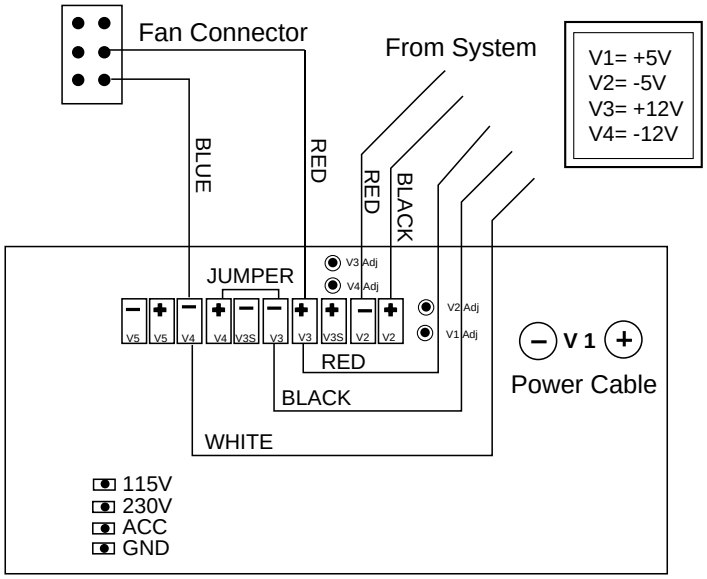


Table 6–3 HC Power Supply Wiring

Connector	Wire	Connection
+V2	Black	Ground from backplane
–V2	Red	–5 V to backplane
+V3S	—	Blank
+V3	Red	+12 V to backplane
	Red	To fan tray
–V3	Black	Ground to backplane for ±12 V
	Black	Jumper to +V4
–V3S	—	Blank
+V4	Black	Jumper from –V3
–V4	White	–12 V to backplane
	Blue	To fan tray
+V5	—	Blank
–V5	—	Blank

6.5 Replacing the DECmpp 12000/Sx DPU Fan Tray

Take the following steps to replace the DECmpp 12000/Sx DPU fan tray:

1. Open the front and rear doors.
2. Turn off the DPU. Set the 30 A circuit breaker OFF, and disconnect the power cord from the power source.
3. From the back of the DPU, unplug the power cable that runs from the power tray to the fan tray.
4. Using a 5/16-inch open-end wrench and a 5/16-inch socket wrench, remove the +5 V bus bars from the +5 V power supply. Gently pull them forward, and lay them down so that they are out of the way of the fan tray when it is pulled out.
5. Remove the four Phillips Screws fastening the fan tray to the enclosure.
6. Pull the fan tray out smoothly. Be careful not to pinch the +5 V cables. This can break the insulation, causing the +5 V power to short to ground.
7. Note the arrow indicating the tray's proper orientation. This is important when installing the new fan tray.
8. Slide the new fan tray in. Be careful not to pinch the +5 V cables. This can break the insulation, causing the +5 V power to short to ground.
9. Replace the four Phillips Screws, securing the fan tray to the frame.
10. Connect the power cable to the fan tray.
11. Connect the +5 V bus bars. Make sure that the bus bars are not shorting to the frame or to each other.
12. With an ohmmeter, check the resistance between +5 V and ground. A zero ohm reading indicates a short which must be investigated.
13. Turn on the DPU, and verify that the new fan is operating properly.

6.6 Replacing the DECmpp 12000–LC/Sx DPU Fan Tray

Take the following steps to replace the DECmpp 12000–LC/Sx DPU fan tray:

1. Turn off the system power, as described in Chapter 1. Set the 15 A circuit breaker to OFF (down), and unplug the power cord from the power source.
2. Open the front and lower-front doors.
3. Open the front inner door.
4. Remove the front inner door:
   - Unplug the switch connector from the lightpipe (under the top cover).
   - Pull down on the latch on the upper hinge and pull the door out. Place the door upright against the wall.
5. Disconnect the power connector from the fan tray.
6. Remove the four Phillips Screws that secure the fan tray to the frame. In some cases, to be able to slide the fan tray out, you may need to remove the +5 V cables from the power supply.
7. Slide the fan tray out.
8. Slide in the replacement fan tray. Reconnect the +5 V cables to the power tray if they were removed.
9. Replace the four Phillips Screws that secure the fan tray to the frame.
10. Connect the power connector to the fan tray.
11. Replace the front inner door:
    - While holding the upper hinge latch down, put the door in place, securing the lower hinge and then sliding the upper hinge into place so that the latch connects properly.
    - Reconnect the switch connector to the rightmost lightpipe connector.
12. Close and latch the front inner door.
13. Close and latch the front and lower-front doors.
14. Turn on the power to the DPU and make sure that the fan is working properly.

6.7 Removing and Replacing the Lightpipe PCB

The DECmpp 12000 and DECmpp 12000–LC DPUs use identical lightpipe PCBs. However, the bracketing and mounting locations are different and the method to access them differs somewhat.

6.7.1 DECmpp 12000/Sx Lightpipe

In the DECmpp 12000 DPU, the lightpipe is mounted to the inside surface of the front door. Follow these steps to remove and replace the lightpipe.

1. Turn the DPU circuit breaker OFF.
2. Open the DPU front door. The lightpipe PCB is mounted to the inside surface of the front door.
3. Disconnect the lightpipe 50-pin ribbon signal cable, control panel cable, and power cable.
4. Loosen but do not remove the four screws that secure the lightpipe PCB bracket to the door.
5. Lift the lightpipe PCB bracket up and off the mounting studs and screws.
6. Put the replacement lightpipe PCB into position over the mounting studs and screws, checking the alignment of the lightpipe and the lightpipe cutouts in the door.
7. Secure the bracket of the replacement PCB to the door by turning the four screws. Do not tighten; the PCB should be able to be moved slightly.
8. Connect the lightpipe power supply cable, control panel cable, and the 50-pin ribbon signal cable to the lightpipe PCB.
9. Turn the PDA circuit breaker ON.
10. Place the lightpipe LED test switch in the TEST position.
11. Fine tune the alignment of the lightpipe and the cutouts in the door to ensure maximum light transmission through the cutouts.
12. Tighten the four screws securing the PCB bracket to the door.
13. Place the lightpipe LED test switch in the NORMAL position.
14. Close the DPU front door.

6.7.2 DECmpp 12000–LC/Sx Lightpipe

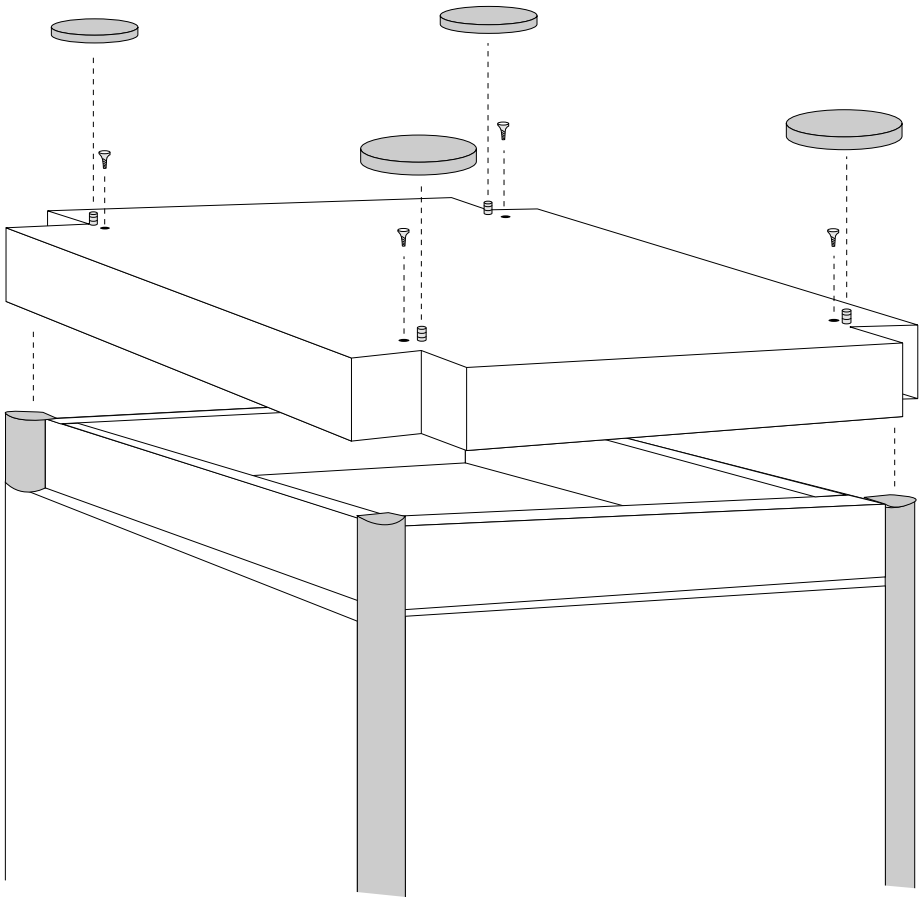
In the DECmpp 12000–LC DPU, the top of the enclosure must be removed to access the lightpipe. Follow these steps to remove the top of the -LC DPU and replace the lightpipe.

1. Turn the DPU circuit breaker OFF (down).
2. Open the DPU front door.
3. Unscrew the corner caps on the MPDA top (Figure 6–10).
4. Remove the four screws attaching the top.
5. The bottom edge of the lightpipe printed circuit board (PCB) is visible under the top-front edge. Disconnect the ribbon signal cable connector, the 9-pin control panel connector, and the 9-pin power connector.
6. Turn the enclosure top over, and place it on a flat surface.
7. Remove the four screws attaching the lightpipe PCB to the top inner edge (Figure 6–11).
8. Check the alignment of the replacement PCB LEDs with the lightpipe cutouts.

   If necessary, gently bend the LED wires to align the LEDs properly with the cutouts.
9. Use four screws to attach the replacement PCB to the top.
10. Lift the DPU top, turn it over, and put it in place.
11. Connect the lightpipe power supply cable and the control panel cable, lifting the top front edge up, if necessary.
12. Connect the ribbon signal cable to the lightpipe PCB.
13. Use four screws to attach the DPU top.
14. Screw the corner caps onto the top.
15. Close the DPU front door.
16. Turn the DPU circuit breaker switch ON (up).

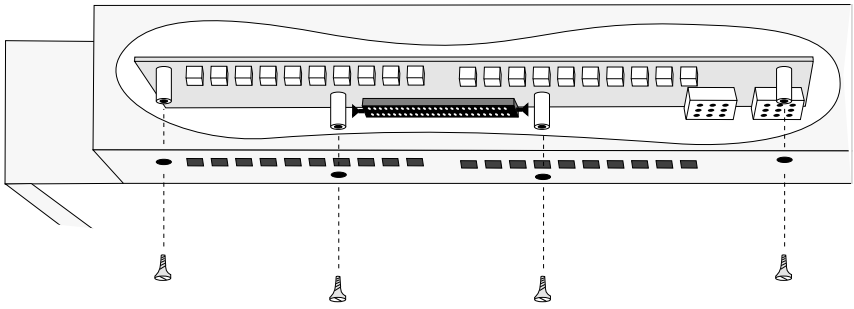
6.7 Removing and Replacing the Lightpipe PCB

Figure 6–10 Removing the DECmpp 12000–LC Enclosure Top



MKV-040000314-60-MPS

Figure 6–11 Replacing the Lightpipe PCB



7

Backplane Jumpers and Upgrading PE Arrays

Chapter 7 explains the various DPU backplane jumpers and describes how to add additional processor element (PE) array printed circuit boards (PCBs) for increased system performance.

7.1 DPU Backplane Jumpers

To understand how to configure the backplane jumpers, you must understand the slot arrangement in the card cage. The DPU card cage is divided into two regions (Figure 7–1).

- The array control unit (ACU), Front-end VME interface (MVIB), and I/O PCB slots
- The PE array/router PCB slots

The DECmpp 12000 backplane has two rows of card cage slots. The top row is for I/O slots, and the bottom row is for PE array/router PCB slots. Looking at the backplane slots from the DPU front, the ACU is in the upper-right I/O slot. The second I/O slot from the right is always occupied by the Front-end VME interface (MVIB). This is a T6000 VMEbus controller PCB mounted on a Parallel VME 6U Adapter PCB. The balance of the I/O slots are reserved for the PVME controller and I/O interface PCBs such as that used for the disk array.

The bottom row has 1, 2, 4, 8, or 16 PE array PCBs in contiguous locations starting from the right and router PCBs in slots not containing PE array PCBs.

Note

When you are looking into the card cage, this view is reversed.

The DECmpp 12000–LC card cage has one row of 11 card cage slots. Looking at the backplane slots from the DPU front and starting from the right, the first slot is the ACU PCB slot, the second is the front-end VME interface PCB, the next five are I/O slots, and the remaining four are PE array/router PCB slots.

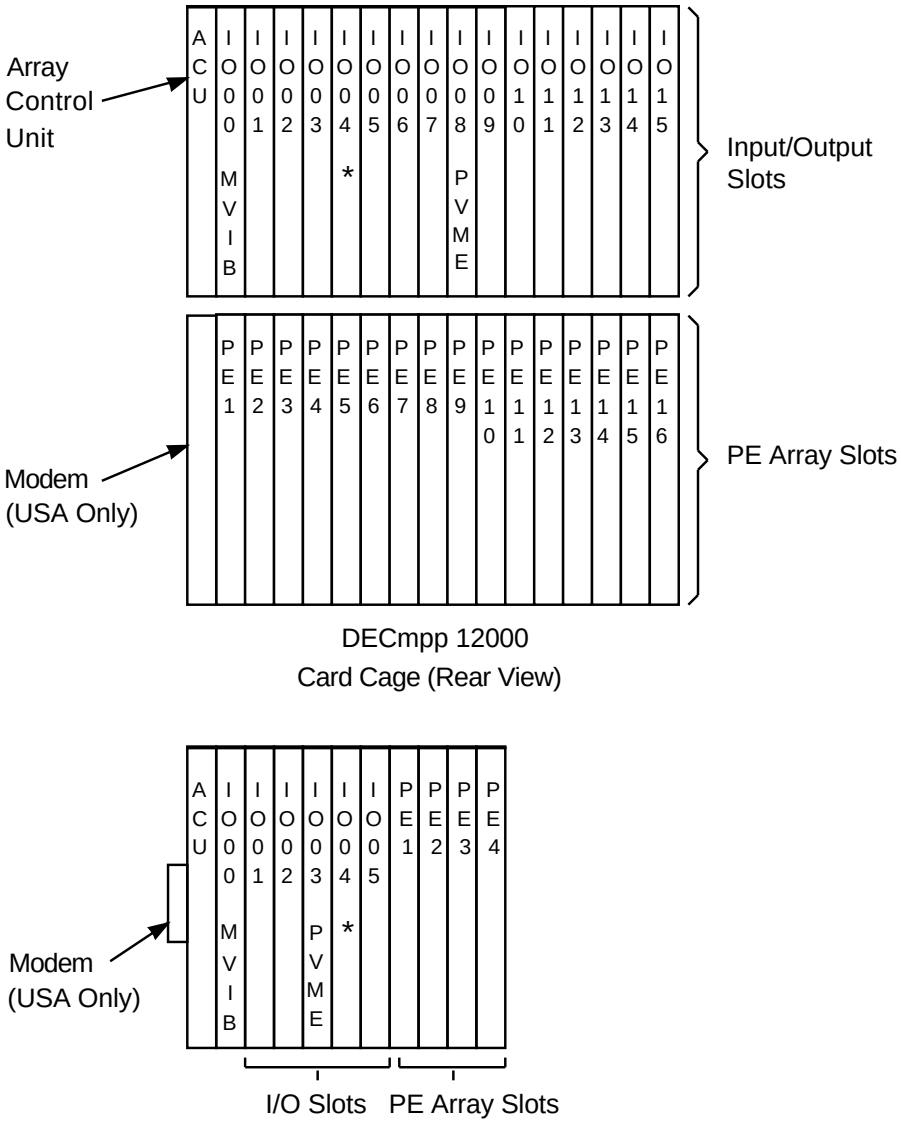
The DECmpp 12000–LC can have 1, 2, or 4 PE array PCBs at contiguous locations and router PCBs in slots not containing PE array PCBs.

Note

When you are looking into the card cage, this view is reversed.

7.1 DPU Backplane Jumpers

Figure 7–1 DPU Card Cage Slots



* Slot IO04 Reserved for Optional PDA Interface

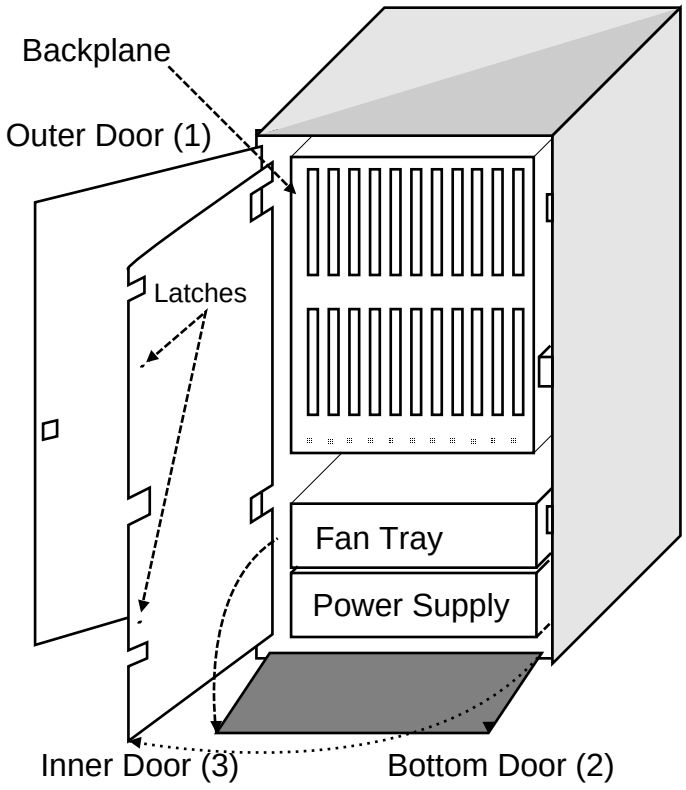
MKV−040000314−03−RAGS

7.1.1 Backplane Access

Backplane access is through the front of the DPU. DECmpp 12000 enclosures have a conventional front door which is opened by unlocking a 1/4-turn Allen head cap latch. DECmpp 12000–LC enclosures have three doors on the front of the DPU, as shown in Figure 7–2.

1. Open the outer top and bottom doors.
2. Open the inner door, using the door latch key.
3. Reverse these steps to close the DPU front doors: close the inner door first, then the bottom door, and finally the outer door.

Figure 7–2 DECmpp 12000–LC/Sx Front Doors



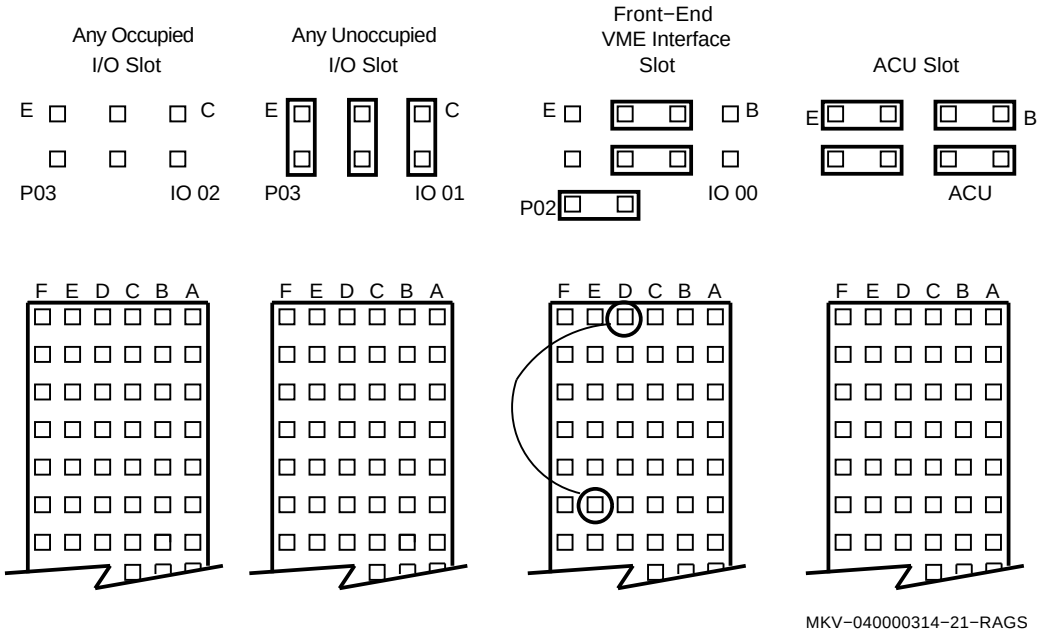
7.1.2 ACU, VMEbus, and I/O Jumpers

As shown in Figure 7–3, there are jumpers at the top of the backplane for the ACU, IO00 (MVIB), and each I/O slot (two shown). The jumpers over the ACU and IO00 slots should be installed exactly as shown. As MPVMEbus I/O devices are added and removed, corresponding jumpers above each I/O slot must be added and removed.

Each I/O slot, I/O01 through I/O14, has a set of pins above it which allow certain MPVMEbus signals to be jumpered past those slots which are not occupied by PCBs. Going from right to left, ensure that the set of three jumpers have been removed from above all I/O01 through I/O14 slots that are occupied by MPVMEbus PCBs.

Conversely, ensure that jumpers are installed above all unoccupied I/O slots up to, but not including, the slot containing the last I/O PCB. Slots after (to the left of) the last occupied I/O slot may or may not have the jumpers installed. It is of no consequence.

Figure 7–3 DPU Backplane Jumpers



7.1.3 X-Net Jumpers

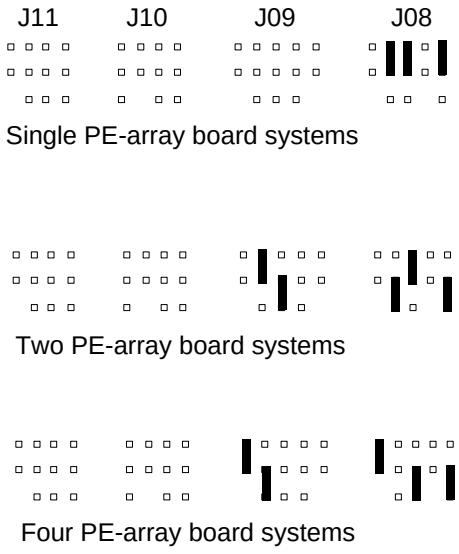
The X-Net jumpers control the X-Net connections between PE array PCBs. They are on the bottom of the backplane, beneath PE array slots 0 through 15 on DECmpp 12000 systems, and beneath slots 0 through 3 on DECmpp 12000–LC systems.

When changing X-Net jumpers:

- Only slots with PE array PCBs should have X-Net jumpers installed. Never install any X-Net jumper on a slot that does not contain a PE array PCB.
- The X-Net jumper installations for all occupied PE array PCB slots can change when the number of PE array PCBs in the system changes. When you increase or decrease the number of PE array PCBs, review the X-Net jumper configurations for every slot containing a PE array PCB.

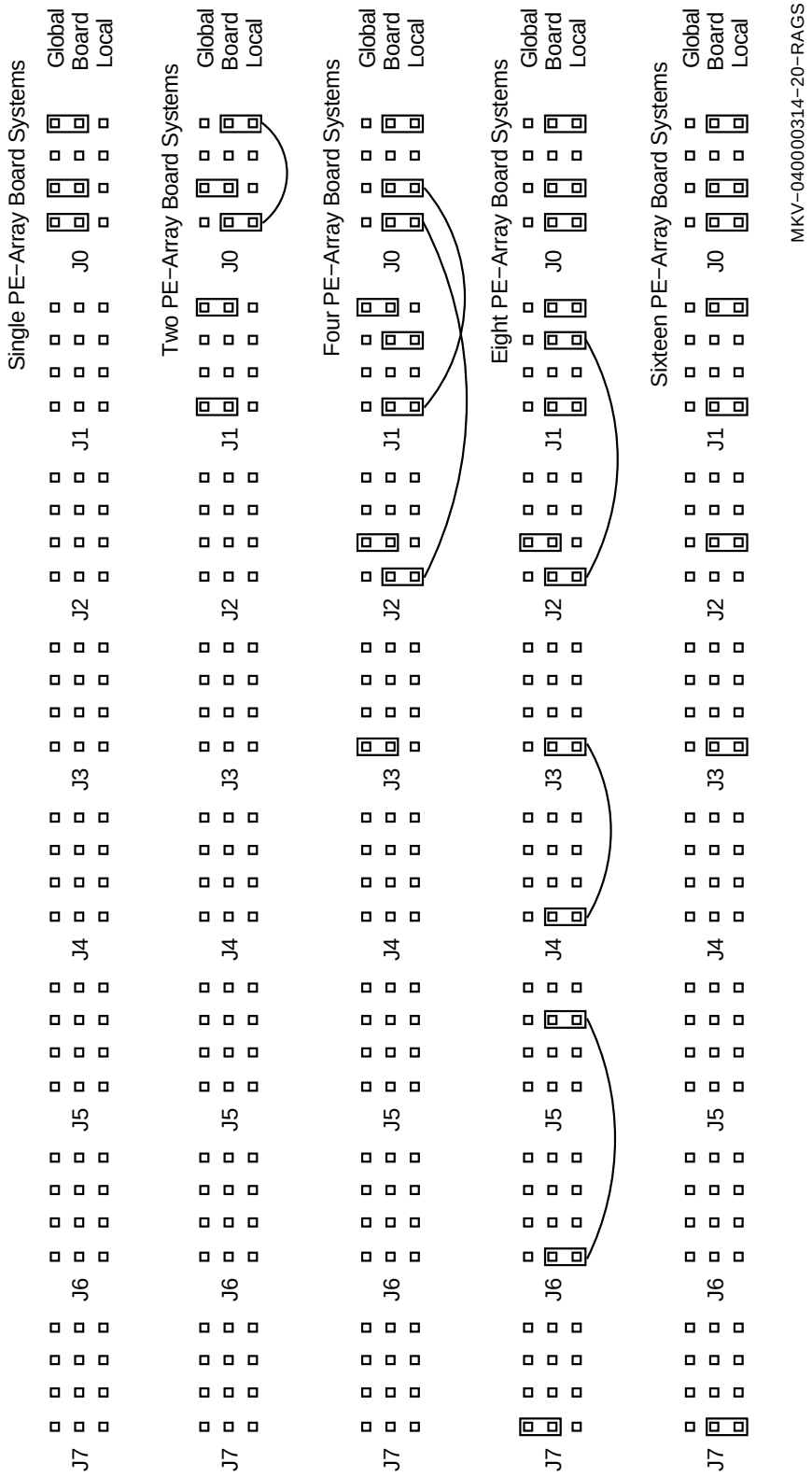
Figure 7–4 shows jumpering for DECmpp 12000–LC configurations. Figure 7–5 shows the X-Net jumpering for DECmpp 12000 configurations.

Figure 7–4 DECmpp 12000–LC/Sx X-Net Jumper Configurations



7.1 DPU Backplane Jumpers

Figure 7–5 DECmpp 12000/Sx X-Net Jumper Configurations



7.2 System Issues for Upgrades

Note

Observe the following configuration rules:

- Every DECmpp system must have the correct number of supported PE array PCBs. DECmpp 12000 systems may have 1, 2, 4, 8, or 16 PE array PCBs. DECmpp 12000–LC systems may have 1, 2, or 4 PE array PCBs. No other combinations are supported.
- When you add PE array PCBs, you must change the data parallel unit (DPU) backplane X-Net jumpers.
- PE array PCBs start in the first PE array PCB slot and occupy contiguous slots up to the maximum installed. Each remaining PE array PCB slot must have a router PCB installed. No PE array PCB slot can be left empty.

No system software changes are required. The system software automatically reconfigures itself to accommodate any supported number of PE array PCBs when booted.

With one exception, DPU programs run on any legal size PE array (1, 2, 4, 8, or 16) and increasing the array size increases the run speed. However, if you have hard-coded the size of the processor array in a DECmpp Programming Language (MPL) program, the program may need to be changed and recompiled before being run on a different-sized PE array.

7.3 Adding Processor Element Array PCBs

Follow this procedure to add additional PE array PCBs. Refer to the guidelines in Section 6.3 for instructions on handling PCBs and Figure 7–1 for PCB placement in the DPU card cage.

Caution

Always wear antistatic straps. PE array and router PCBs can be damaged very easily by an inadvertant static discharge.

1. Open the DPU rear door to access the card cage.
2. Turn off the power to the DPU as described in Chapter 1.
3. Identify the slots that will receive the new PE array PCBs.
4. Remove the router PCBs from these slots:
   a. Remove the lower PCB retainer bar.
   b. Loosen the captive screws (if present) at the top and bottom of the PCB faceplate.
   c. Use the ejector levers to release the PCB and move it outward from its slot.
   d. Place the router PCB on an antistatic mat.
5. Carefully slide the new PE array PCB into the slot. When it is almost in place, you might encounter some resistance; push firmly to seat it in the backplane connectors.
6. Secure the PCB to the card cage by closing the ejector levers.
7. Tighten the captive screws (if present) at the top and bottom of the PCB faceplate.
8. Place the router PCB just removed in the PE array packing material.
9. Repeat for each new PE array PCB.
10. Replace the lower PCB retainer bar.
11. Open the DPU front door and inner front door (-LC only).
12. Reconfigure the backplane X-Net jumpers, as described in Section 7.1.3.
13. Turn on the system, as described in Chapter 1.
14. Close and lock the DPU rear door, the DPU inner front door (-LC only), and the front door.
15. Run the PE diagnostics as a confidence check.

7.4 Reconfiguring Processor Element Array PCBs

PE arrays can be downgraded so that a DECmpp system can be used with reduced performance while PE array spares are being acquired. However, this necessitates additional router PCBs because no PE array slot may be left unoccupied. The process of downgrading a PE array is very similar to that of upgrading.

Caution

Always wear antistatic straps. PE array and router PCBs can be damaged very easily by an inadvertant static discharge.

1. Open the DPU rear door to access the card cage.
2. Turn off the power to the DPU as described in Chapter 1.
3. Determine the next lower supported PE array configuration (Section 7.2).
4. Identify the slots that will have PE array PCBs taken out and router PCBs installed.
5. Remove the PE array PCBs from these slots:
   a. Remove the lower PCB retainer bar.
   b. Loosen the captive screws (if present) at the top and bottom of the PCB faceplate.
   c. Use the ejector levers to release the PCB and move it outward from its slot.
   d. Place the PE array PCB on an antistatic mat.
6. Carefully slide the spare router PCB into the slot. When it is almost in place, you might encounter some resistance; push firmly to seat it in the backplane connectors.
7. Secure the PCB to the card cage by closing the ejector levers.
8. Tighten the captive screws at the top and bottom of the PCB faceplate.
9. Repeat for each PE array PCB being removed.
10. Replace the lower PCB retainer bar.
11. Open the DPU front door and inner front door (-LC only).
12. Reconfigure the backplane X-Net jumpers to support the temporary configuration, as described in Section 7.1.3.
13. Turn on the system, as described in Chapter 1.
14. Close and lock the DPU rear door, the DPU inner front door (-LC only), and the front door.
15. Run the PE diagnostics as a confidence check.

As soon as the PE array spares arrive, reconfigure the array to the original state (using the new spares) and rerun PE diagnostics as a confidence check.

A

Recommended Spares List

Table A–1 is a recommended spares listing (RSL) for the DECmpp 12000/Sx and DECmpp 12000–LC/Sx Series Data Parallel Units.

Table A–1 DECmpp 12000/Sx Data Parallel Unit RSL

DEC PN	Vendor PN	Description
29-29371-01	3400-0000-01	Array control unit PCB[1,2]
29-29397-01	3400-0002-00	1K PE PCB w/16MB RAM[1,2]
29-29396-01	3400-0002-01	1K PE PCB w/64MB RAM[1,2]
29-29372-01	3400-0018-00	Router PCB [1,2,3]
29-29377-01	3400-0031-02	Parallel VME I/O controller PCB[1,2]
29-29563-01	4200-0079-00	100-pin front-end VME I/O cable[1,2]
29-29564-01	4200-0078-00	RS-232 modem cable[1,2]
54-20087-01	N/A	VMEbus interface (MVIB)[1,2]
54-20085-01	N/A	TURBOchannel VMEbus controller (3VIA)[1,2]
12-25537-01	N/A	VME loopback connector[1,2]
29-29373-01	8000-0009	Modem assembly (for United States systems only)[2]
70-30318-01	N/A	Lightpipe assembly[2]
70-30319-01	N/A	DPU power supply assembly[2]
70-30317-01	N/A	DPU fan cage assembly[2]
FC-10169-AC	N/A	VME test module[4]

[1]Branch-level spare

[2]Geography-level spare

[3]Recommend sufficient quantity be stocked (see Product Support Plan)

[4]Special test tool

B

Data Parallel Unit Reference Pages

This appendix contains the following Data Parallel Unit reference pages.

- acu_ppdma(1)
- mpconfig(1)
- mpi(1)
- mpq(1)
- mpstat(1)
- pe_arith(1)
- pe_ckonet(1)
- pe_diag(1)
- pe_func(1)
- pe_macro(1)
- pe_memdiag(1)
- pe_rtbp(1)
- pe_rtdiag(1)
- pe_rtr(1)
- pe_scan(1)
- dpumanager(8)
- mpshutdown(8)

acu_ppdma(1)

acu_ppdma — DECmpp Sx array control unit (ACU) Peek/Poke, DMA transfer test

Syntax

acu_ppdma [-qtb]

Description

The acu_ppdma command tests the ACU board's ability to perform a DMA transfer between the front-end processor's memory and PMem. It also exercises the peek /poke capability.

Options

-b
Use this option to select the Burn-in test; runs the diagnostic repetitively, reporting the error count at the end of each pass.

-q
Use this option to select the Quick test; selects a brief version of some of the tests.

-t
Use this option to select the Terse message style; the test prints only the most essential messages.

Diagnostics

The test defines an array of 32 x 32 processor elements for its area of operation, starting at the processor element at row 0, column 0. It loads a buffer in front-end processor memory with 16K bytes of random data which it then transfers via DMA to PMem, allotting 16 bytes to each processor element (PE). Then, by means of the peek/poke mechanism, it reads the data a byte at a time from PMem and verifies its correctness.

Next, the test loads a new random sequence of data a byte at a time into PMem. It then makes a DMA transfer from PMem into the front-end processor's memory where it checks the correctness of the data.

If the test detects a data error, it determines where it occurred in the PE array and reports this information to the user. If too many errors occur, the test is aborted.

Files

$MP_PATH/field/bin/acu_ppdma

mpconfig(1)

mpconfig — DECmpp Sx data parallel unit (DPU) configuration information, Version 1.1

Syntax

mpconfig

Description

The mpconfig(1) command prints out information about the DPU. This command can only be run from a machine attached to a DECmpp Sx DPU. The information generated is similar to the following:

```
DECmpp Sx DPU Model MP-1204 (64 rows, 64 columns)
Serial number: 0
Microcode version: 2.2.67
Hardware option: 3
processor element (PE) memory size: 16384 bytes
array control unit (ACU) memory size: 114688 bytes
```

Files

/usr/tmp/.dpuconfig

mpi(1)

mpi — DECmpp Sx data parallel unit (DPU) configuration information, network style, Version 1.1

Syntax

mpi [*hostname*]

Description

The mpi command prints out information about all data parallel units (DPUs) on any attached local area network supporting SO_BROADCAST sockets and the broadcast address INADDR_BROADCAST. The broadcast message is received by a daemon program, maspard(8), running on all machines that have a DPU attached. Upon receipt of the broadcast message, each daemon sends a message describing the DPU attached to the client program, mpi. The mpi can be run on any machine in the network; even those without a DPU. The information generated is similar to the following:

```
Copyright (c) 1991 MasPar Computer Corporation. All rights reserved.
                            processor
                            element
Version 2.2 MACHINE  TYPE     (PE)      PMem   CMem    MODEL UCODE   QUEUE
            svsales  4.2RISC  64x64     65536  114688  1104  2.4.11    0
            krusty   4.2RISC  32x32     16384  114688  1101  2.7.8     0
            pigpen   4.2RISC  64x64     16384  114688  1204  2.6.228   0
            lucy     4.2RISC  64x128    16384  114688  1208  2.6.179   1
            linus    4.2RISC  32x32     16384  114688  1101  2.7.8     0
            alpha2   4.2RISC  32x64     16384  114688  1202  2.6.238   1
```

Files

/usr/tmp/.dpuconfig

See Also

mpq(1)

mpq(1)

mpq — DECmpp Sx job queue examination program, Version 1.1

Syntax

mpq [hostname]

Description

The mpq command examines the shared memory segment maintained by dpumanager(8) that contains the list of DECmpp Sx data parallel unit (DPU) jobs waiting for execution. When a host name is specified, information on that machine's job queue is given. When no host name is specified, the job queue for the current machine is shown.

For each job queued, mpq reports the current rank in the queue, the user's name, the id of the process that opened the DPU device file, the process group, the processor element (PE) memory size requirement, time in current status (active or waiting), and the current job status.

The job status may be one of these values:

- active — The job currently holds the DPU device.
- waiting — The job is waiting for DPU to be available.
- inactive — The DPU device has been opened by the job but access has not yet been requested (or the device has been released).

Jobs in DPU memory that are not actually running have the additional status of swapped. An example of an inactive job is a program that has run to completion under the symbolic debugger and has not been quit or restarted yet. A job that is inactive does not hold the DPU device; other jobs can use the DPU while that job is still inactive.

If dpumanager(8) is not running, no output is given.

Restrictions

Due to the dynamic nature of the queue, mpq may occasionally print erroneous information.

See Also

dpumanager(8), mpstat(1)

mpstat(1)

mpstat — Prints DECmpp Sx job accounting statistics, Version 1.1

Syntax

mpstat [*options*]

Description

The mpstat command examines, and optionally clears, the accounting file generated by the dpumanager(8) program. Depending on the options specified, it prints a list of all jobs run and/or a summary.

Options

-a
Use this option to request a chronological list of all of the relevant jobs listed in the accounting file. The job starting time, user name, process id, running time and waiting time is printed for each job. Waiting time includes time waiting for access, time when a job is swapped out, inactive periods after the job has released the data parallel unit (DPU) but has not closed the device, and a small amount of system overhead.

-c
Use this option to request the accounting file to be cleared at the end of the program.

-f *file*
Use this option to specify the name of an accounting file to be used in place of the default.

-s
Use this option to request a summary showing the number of jobs, the average and maximum running times, and the average and maximum wait times. This is the default when neither the -a nor -c options are specified.

-t *job_count*
Use this option to specify that only the tail *job_count* entries in the accounting log are of interest. This allows you to see only the most recent log entries. Note that other filters (like the -u option) are applied after this one, so you may see less than the specified number of entries. This option may be used with both the -a and -s options.

-u *userName*
Use this option to cause mpstat to only count jobs for the specified user (login name).

Restrictions

The accounting file may only be cleared when you are running mpstat as root.

Files

/usr/adm/dpuacct

See Also

dpumanager(8), mpq(1)

pe_arith(1)

pe_arith — DECmpp Sx processor element (PE) arithmetic operations test

Syntax

pe_arith [-bqt]

Description

The pe_arith command tests the operation of the arithmetic commands.

Using peek/poke, the front-end program loads four buffers of data into each PE's PMem: an 8-bit, 16-bit, 32-bit and 64-bit buffer.

The front-end program starts the back-end program and gives it the op code and op size. The back-end program has each PE perform the requested operation, each PE storing the result in another data buffer in its PMem. The back-end program is halted and the front-end program directly checks the results in PMem, using peek/poke.

Options

-b
Use this option to specify the Burn-in test; runs the diagnostic repetitively, reporting the error count at the end of each pass.

-t
Use this option to specify terse message style; prints only the most essential messages.

Files

$DIAG_PATH/field/bin/pe_arith

pe_ckonet(1)

pe_ckonet — DECmpp Sx octagon net test

Syntax

pe_ckonet [-qtb]

Description

The pe_ckonet command tests the ability of processor elements (PE) to shift data using the xnet. The test takes a 32-bit pattern and does a zero distance 32-bit xnet move north, northeast, east, southeast, south, southwest, west, and finally northwest. It compares the value received by northwest with the value originally sent. It repeats the test, successively increasing the distance, until data is shifted a distance of 2048.

If an error occurs, the test reports the size of the octagon loop (the distance shifted) encountering the error. Nothing further can be inferred about the location of the fault. However, when this happens, the test then uses peek/poke to check each PE for parity error. It reports the exact location of the offending PE. Although it is possible that some other failure mechanism could cause a data errror in the octagon shift ring, it is most likely to be a parity error.

Options

-b
Use this option to specify the Burn-in test. This runs the diagnostic repetitively, reporting the error count at the end of each pass.

-q
Use this option to specify quick test. This selects a brief version of some of the tests.

-t
Use this option to specify terse message style. The test prints only the most essential messages.

Files

$MP_PATH/field/bin/pe_ckonet—Executable binary

pe_diag(1)

pe_diag — DECmpp Sx processor element (PE) board and backplane diagnostic

Syntax

pe_diag [-qtb]

Description

The pe_diag command performs the following diagnostics:

1. Serial Scan Tests (pe_scan) — Tests the serial scan chains on the array control unit (ACU) board, the PE board and the router boards:
   - ACU Board
     - Main scan chain
     - EEPROM scan chain
   - PE and ROUTER boards
     - Router shift chain S1, S2 and S3
     - EEPROM scan chain
     - GOR scan chain
     - PReg scan chain
     - Parity: PReg address parity, PE instruction parity, RT instruction parity, M-Machine instruction parity
2. Macro Instruction Tests (pe_macro)
   - Arithmetic and logic instruction tests
   - Move instruction tests
   - Load/store instruction tests
   - XNet instruction tests
   - Router instruction tests
   - Full backplane router instruction tests
3. XNet Tests (pe_xnet)
   - XNet simple test
   - XNet complex test
   - XNet pipe mode simple test
   - XNet pipe mode complex test
   - XNet copy mode simple test
   - XNet copy mode complex test
   - PE parity test
4. Router backplane test (pe_rtbp)

Options

-b
Use this option to specify the Burn-in test; runs the diagnostic repetitively, reporting the error count at the end of each pass.

-q
Use this option to select quick test; selects a brief version of some of the tests.

-t
Use this option to specify terse message style; prints only the most essential messages.

Files

Executable file found in directory `$MP_PATH/field/bin`:
`pe_diag`.

See Also

`pe_macro`, `pe_rtbp`, `pe_scan`, `pe_xnet`

pe_func(1)

pe_func — DECmpp Sx processor element (PE) function test

Syntax

pe_func [-t]

Description

The pe_func command enables the entire array and has each PE perform the following functions:

- div32: 0x12345678 / 1 = ?
- mul64: 0x123456789abcdef0 * 0xfedcba9876543210 = ?
- div32: 0x11111111 / 2 = ?
- div32: 0x87654321d / 1 = ?
- div64: 0x123456789abcdef0 / 1 = ?
- udiv32: 4 / 2 = ?
- add8: 0xff + 0xff = ?
- add8: 0xff + 1 = ?
- add8: 0 + 1 = ?
- add8: 1 + 1 = ?
- add8: 2 + 2 = ?
- add8: 4 + 4 = ?
- add8: 0x10 + 0x10 = ?
- add8: 0x20 + 0x20 = ?
- add8: 0x40 + 0x40 = ?

A GOR of the answers to each function is returned to the front end and verified. In case of error, prints the operation number, the expected result and the actual result.

Options

-t
Use this option to specify terse message style; prints only the most essential messages.

Files

Binary executable file: $DIAG_PATH/field/bin/pe_func

pe_macro(1)

pe_macro — DECmpp 12000/Sx processor element (PE) board macro instruction tests

Syntax

pe_macro [-qtb]

Description

The pe_macro command performs the following tests:

1. Arithmetic and logic instruction tests
   - ortest1 — Global OR test
   - ebit1 — E-bit operation test
   - And1Tests1 — Single bit AND test
   - AndTests1 — Preg to PReg AND test
   - AndTests2 — Immediate to PReg AND test
   - AndTests3 — 64-bit AND test
   - OrTests1 — Register to register OR test
   - OrTests2 — Immediate to register OR test
   - sadd1 — Short add test
   - AddTests1 — Register to register ADD test
   - AddTests2 — Immediate to register ADD test
   - AddTests3 — Check flags and ADD and MOV instructions
   - SubTests1 — Preg to PReg subtraction test
   - SubTests1 — Immediate to PReg subtraction test
   - ShiftTests — Shift instruction test
   - mult1 — Multiply test
   - div1 — 64-bit signed integer divide test
   - divu0 — 64-bit unsigned integer divide test
   - fadd1 — 64-bit floating point add test
   - fmult1 — 64-bit floating point multiply test
   - random1 — Random macro instruction test
2. Move instruction tests
   - MoveTests1 — 8-bit move test
   - MoveTests2 — 16-bit move test
   - MoveTests3 — 64-bit move test
   - MoveTests4a — MOV1: set flag bits test (part 1)
   - MoveTests4b — MOV1: set flag bits test (part 2)

- MoveTests4c — MOV1: clear flag bits test (part 1)
- MoveTests4d — MOV1: clear flag bits test (part 2)
- MoveTests4e — MOV1: PReg to flag bit test
- MoveTests4f — MOV1: Immediate to PReg test
- MoveTests5a — MOV1UC: Flag to flag 1's test (part 1)
- MoveTests5b — MOV1UC: Flag to flag 1's test (part 2)
- MoveTests5c — MOV1UC: Flag to flag 0's test (part 1)
- MoveTests5d — MOV1UC: Flag to flag 0's test (part 2)
- MoveTests5f — MOV1UC: Immediate to PReg test
- Else1ucTests — ELSE1UC instruction test
- MoveTests6 — 8-bit & 32-bit move test

3. Load/store instruction tests
   - msol0 — Directly addressed solitary load/store test
   - msol1 — Indirectly addressed solitary load/store test
   - msol2 — Indirectly addressed single PE load/store test
   - LdSt — Load/store instruction test
   - mtest0 — 32-bit direct load/store test
   - mtest1 — Load, store and M-bit queue test
   - mtest2 — Indirect load/store test
   - mtest3 — Simple indirectly addressed load/store test
   - mtest4 — 64-bit directly addressed load/store test
   - mtest5 — 32-bit byte offset load/store test
   - tagtest0 — Tag stall test - many PReg's to single PMem location
   - tagtest0f — Tag stall test - many PReg's to many PMem locations
4. XNet instruction tests
   - xnetNW — XNet shift NorthWest test
   - xnetN — XNet shift North test
   - xnetNE — XNet shift NorthEast test
   - xnetE — XNet shift East test
   - xnetSE — XNet shift SouthEast test
   - xnetS — XNet shift South test
   - xnetSW — XNet shift SouthWest test
   - xnetW — XNet shift West test
   - xnet1 — Shift data from odd to even columns and back
   - onet — XNet octagon shift test
   - onet1 — XNet octagon shift test (with PMem activity)

5. Router instruction tests
   - `rt0a` — 16-bit router send test
   - `rt0b` — 16-bit router fetch test
   - `rt0c` — Router send test (1, 8, 16, 32, 64 bits
   - `rt0d` — Router fetch test (1, 8, 16, 32, 64 bits)
   - `rt0e` — Router send and fetch test
   - `rt0f` — Router send and fetch (while storing) test
   - `rt0g` — Router open/send and fetch/close test
   - `rt1a` — Single PE per cluster router send & fetch test
   - `rt2a` — All PEs send/fetch to self
   - `rt2b` — All PEs r0send/rfetchc to self
6. Full backplane router instruction tests
   - `rt2c` — Forward router send test
   - `rt2d` — Forward router fetch test
   - `rt2e` — Reverse router send test
   - `rt2f` — Reverse router fetch test
   - `rt2g` — All PEs send/fetch to self (M-machine busy)

Basic Test Strategy

Each of the following tests employ the same structure and control method. Since it is nearly impossible for a short error message to convey the full context including the events preceding the error, it is important that you be able to trace this back using the program listings.

The diagnostic consists of a common front-end program (HDB script) which downloads and starts the assembly language program (macro program), which runs on the array control unit (ACU). The macro program performs various operations and returns results to the front end program using the FRBEQ. The front-end program has access to a golden file which contains all the correct values to be returned using the FRBEQ.

The golden file also contains an error message with each value and when the actual value does not compare, the front-end program prints the error message as well as the error message number.

If you want to delve more deeply into the matter, refer to the macro program listing. This listing contains a comment at each statement sending data down the FRBEQ. These comments are numbered, and the number corresponds to the error message number. Knowing this, you can quickly locate the place in the macro program where the error occurred.

Sometimes when testing a faulty system, a FRBEQ timeout is a common problem. The front-end program loads and starts the macro program, then waits for data to appear on the FRBEQ. If none appears after a reasonable length of time, the front-end program aborts the test. However, it also reports the contents of the Halt Code Status Register, which tells why the macro program halted. This can give a clue to the difficulty.

Sometimes the macro program sends data out the FRBEQ for awhile before halting. The front-end program reports which record it was waiting for when it timed out, and this information may prove useful.

Individual Test Descriptions

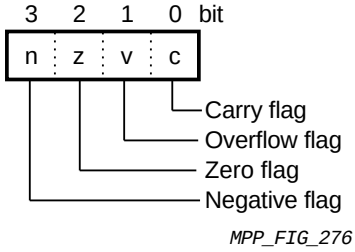
The following list describes the individual tests in alphabetical order:

- `AddTests1`: Register To Register ADD Test — This test performs 8-bit, 16-bit, 32-bit, and 64-bit addition between PRegs. It performs the following sums for each of the four word sizes:
  - 1 + 1 = 2
  - 1 + –1 = 0
  - –1 + –1 = –2
  - 1 + –81 = –80

  It demonstrates the ability to handle both positive and negative numbers, as well as the proper sign extension for each word size.

  The test enables all the PEs and loads an identical set of data into each PE's PRegs. After each addition, the test performs a global OR of each PE's answer. If it detects an error, the test prints an error message which gives the addition the test was attempting, the expected answer and the actual answer.

  After each addition, it performs a global OR of each PE's flag bits. If any of these is in error, an error message is printed giving the operation attempted, the expected flag bits and the actual flag bits. The flag bits are collected as shown:



MPP_FIG_276

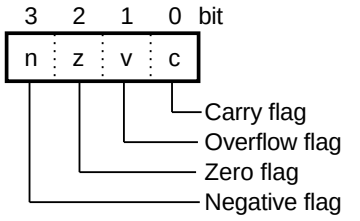
  If this test reports a wrong answer, it indicates that the fault is associated with one or more PEs.

- `AddTests2`: Immediate To Register ADD Test — This test performs 8-bit, 16-bit, 32-bit, and 64-bit addition between an immediate and a PReg. It performs the following sums for each of the four word sizes:
  - 1 + 1 = 2
  - 1 + –1 = 0
  - –1 + –1 = –2
  - 1 + –81 = –80

  It demonstrates the ability to handle both positive and negative numbers, as well as the proper sign extension for each word size.

The test enables all the PEs. After each addition, the test performs a global OR of each PE's answer. If it detects an error, the test prints an error message which gives the addition the test was attempting, the expected answer and the actual answer.

After each addition, it performs a global OR of each PE's flag bits. If any of these is in error, an error message is printed giving the operation attempted, the expected flag bits and the actual flag bits. The flag bits are collected as shown:



If this test reports a wrong answer, it indicates that the fault is associated with one or more PEs.

- `AddTests3`: Check Flags After ADD And MOV Instructions — This test executes a variety of additions and move instructions.

  The test enables all the PEs and initializes their PRegs with an identical set of data. After each operation, the test does a global OR of each PE's answer, then a global OR of each PE's flag bits.

  If it detects an error in the result, it prints an error message giving the instruction under test, the expected result and the actual result. If any flag is in error, the test prints an error message stating which flag is in error, the expected value and the actual value.

- `And1Tests1`: Single Bit AND Test — This is a test of the AND1 instruction. It uses the following bits as operands:

Flag Bits	PReg's
lsb	2[c4] — where c4 = 1024
lflag	3[c4]
cflag	4[c4]
vflag	5[c4]
zflag	6[c4]
nflag	7[c4]
tflag	8[c4]
fflag	9[c4]
rflag	10[c4]
64[c0]	11[c4]
	12[c4]

The test starts by testing the ability to perform the AND1 function between the following sources and destinations:

SRC	Destination	Message Reference
Each flag bit	lsb	Message #1–#40
Each flag bit	lflag	Message #41–#80
Each flag bit	cflag	Message #81–#120
Each flag bit	vflag	Message #121–#160
Each flag bit	zflag	Message #161–#200
Each flag bit	nflag	Message #201–#240
Each flag bit	tflag	Message #241–#180
Each flag bit	fflag	Message #281–#320
Each flag bit	rflag	Message #321–#360
Each flag bit	64[c0]	Message #361–#400

If it detects an error in the result, it prints an error message giving the instruction under test, the source and destination operands, the expected result and the actual result.

After each AND1 operation, it tests the lflag. If this is wrong, it prints an error message giving the expected and actual value of the flag.

Next, initializing the destinations as shown, the test performs the AND1 function between the following source and destination operands.

Dest Init.	SRC	Destination	Message Reference
1	1025[c2]	Each flag bit	Message #401–#422
1	Each flag bit	Each PReg bit	Message #423–#444
0	1025[c2]	Each flag bit	Message #445–#466
0	Each flag bit	Each PReg bit	Message #467–#488

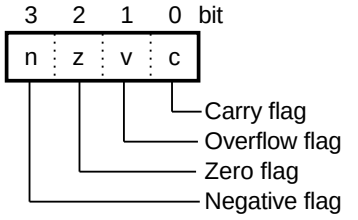
After each AND1 operation, it tests the lflag. If this is wrong, it prints an error message giving the expected and actual value of the flag.

If it detects an error in the result, it prints an error message giving the instruction under test, the destination operand, the expected result and the actual result.

- `AndTests1`: Register To Register AND Test (8, 16, 32 and 64 bit) — This test performs 8-bit, 16-bit, 32-bit, and 64-bit AND operations.

  After each AND, if it detects an error, the test prints an error message which gives the AND the test was attempting, the expected answer and the actual answer.

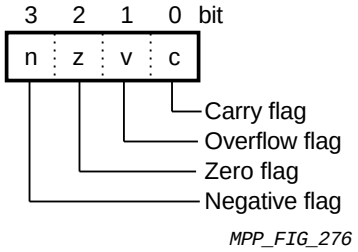
  After each AND, it checks the flag bits. If any of these is in error, an error message is printed giving the operation attempted, the expected flag bits and the actual flag bits. The flag bits are collected as shown:



- AndTests2: Immediate To PReg AND Test (8, 16, 32 and 64 bit) — This test performs 8-bit, 16-bit, 32-bit, and 64-bit AND operations.

  After each AND, if it detects an error, the test prints an error message which gives the AND the test was attempting, the expected answer and the actual answer.

  After each AND, it checks the flag bits. If any of these is in error, an error message is printed giving the operation attempted, the expected flag bits and the actual flag bits. The flag bits are collected as shown:



- AndTests3: 64 Bit AND Test — With all PEs enabled, this test performs eight 64-bit AND operations in immediate succession into eight adjacent PReg locations. Then it checks each of these PReg locations for the correct answer. One purpose of this test is to verify that AND64 operations into adjacent PReg locations do not affect each other.

  This test performs the following PReg to PReg operations:

  – 0x21436587deadbeef & 0x21436587deadbeef –> 0x000[c0]
  – 0x21436587deadbeef & 0x0000000000000000 –> 0x040[c0]
  – 0x21436587deadbeef & 0xffffffffffffffff –> 0x080[c0]
  – 0x21436587deadbeef & 0x1248124812481248 –> 0x0c0[c0]
  – 0x21436587deadbeef & 0xedb7edb7edb7edb7 –> 0x100[c0]

  It then performs the following IMMEDIATE to PReg operations:

  – 0xffffffffffffffff & 0x1248124812481248 –> 0x140[c0]
  – 0xffffffffffffffff & 0xedb7edb7edb7edb7 –> 0x180[c0]
  – 0x0000000000000000 & 0xffffffffffffffff –> 0x1c0[c0]

  After performing all of the above operations, keeping each answer in a different PReg location, the test checks for errors. If it finds an error, it prints an error message giving the attempted AND operation, the expected result and the actual result.

- `Else1ucTests`: ELSE1UC Instruction Test — The ELSE1UC instruction complements the destination value, then ANDs the source with the destination, placing the results both in the destination and in lflag.

  This test uses the following bits as operands:

Flag Bits	PRegs
lsb	2[c4] — where c4 = 1024
lflag	3[c4]
cflag	4[c4]
vflag	5[c4]
zflag	6[c4]
nflag	7[c4]
tflag	8[c4]
fflag	9[c4]
rflag	10[c4]
65[c0]	11[c4]
	12[c4]

  The test starts by testing the ability to perform the ELSE1UC function between the following sources and destinations:

SRC	Destination	Message Reference
Each flag bit	lsb	Message #1–#40
Each flag bit	lflag	Message #41–#80
Each flag bit	cflag	Message #81–#120
Each flag bit	vflag	Message #121–#160
Each flag bit	zflag	Message #161–#200
Each flag bit	nflag	Message #201–#240
Each flag bit	tflag	Message #241–#180
Each flag bit	fflag	Message #281–#320
Each flag bit	rflag	Message #321–#360
Each flag bit	64[c0]	Message #361–#400

  If it detects an error in the result, it prints an error message giving the instruction under test, the source and destination operands, the expected result and the actual result.

  After each ELSE1UC operation, it tests the lflag. If this is wrong, it prints an error message giving the expected and actual value of the flag.

  Next, initializing the destinations as shown, the test performs the ELSE1UC function between the following source and destination operands.

Dest Init.	SRC	Destination	Message Reference
1	1025[c2]	Each flag bit	Message #401–#422

Dest Init.	SRC	Destination	Message Reference
1	Each flag bit	Each PReg bit	Message #423–#444
0	1025[c2]	Each flag bit	Message #445–#466
0	Each flag bit	Each PReg bit	Message #467–#488

After each ELSE1UC operation, it tests the lflag. If this is wrong, it prints an error message giving the expected and actual value of the flag.

If it detects an error in the result, it prints an error message giving the instruction under test, the destination operand, the expected result and the actual result.

- `LdSt`: LOAD/STORE Instruction Test — All PEs are enabled and authorized to make load/store transactions. Each PE stores a constant from PReg to PMem, then loads it back from PMem to (another) PReg:

  ```
  Constant --> PReg --> PMem --> PReg
  ```

  If the test detects an error, it prints an error message showing the chain of store/load operations, and giving the expected and actual result.

  An erroneous value indicates that one or more PEs made an error during this transaction.

- `MoveTests1`: 8-Bit MOVE Test — This test verifies that the MOV8 instruction moves only 8 bits and does not affect adjacent data.

  All PEs are enabled. After initialization, the test makes the sequence of 8-bit moves shown below:

  - Initialization
    - MOV32: 0x5e461237 –> 16[c0]
    - MOV32: 0xfffeffff –> acc
  - Test
    - MOV8: 16[c0] –> 64[c0] –> acc –> 72[c0] = 0x37 (?)

  If it detects an error, it prints an error message giving the operation, the expected result and the actual result.

- `MoveTests2`: 16-Bit & 32-Bit MOVE Test — This test verifies that the MOV16 instruction moves only 16 bits and does not affect adjacent data.

  It enables all PEs, and initializes PReg 16[c0] with a 32-bit constant:

  - Test #1 — Makes three 16-bit moves starting with the data in 16[c0], and verifies the result. Should not move the high order bits from 16[c0].
  - Test #2 — Makes three 32-bit moves starting with the data in 16[c0], and verifies the result. The previous test should not have affected this number.

  If it detects an error, it prints an error message giving the operation, the expected result and the actual result.

- `MoveTests3`: 64-Bit MOVE Test — Assembles a 64-bit number using two 32-bit moves. Moves the 64-bit number using a 64-bit move. Then picks out the low word and high word from the 64-bit number using two 32-bit moves.

- `MoveTests4a`: MOV1: Set Flag Bits Test (part 1) — This is a test of the MOV1 instruction. It uses the following bits as operands:

  lsb
  lflag
  cflag
  vflag
  zflag
  nflag
  tflag
  fflag
  rflag
  64[c0]

  First, it tests that the MOV1 instruction can move an immediate 1 or 0 into each of the above flag bits, and that the lflag is correct after each move (message #1–#41).

  Next, it successively moves a 1 from the source into each of the flag bits in the list shown above.

SRC	Destination	Message Reference
lsb	Each flag bit	#42–#61
lflag	Each flag bit	#62–#81
cflag	Each flag bit	#82–#101
vflag	Each flag bit	#102–#121

  After each move, it checks that lflag is set properly. After moving into all of the flag bits, it checks that each was set properly.

  The remainder of this test is continued in MoveTests4b.ma.

- `MoveTests4b`: MOV1 Set Flag Bits Test (part 2) — This is a test of the MOV1 instruction. It uses the following bits as operands:

  lsb
  lflag
  cflag
  vflag
  zflag
  nflag
  tflag
  fflag
  rflag
  64[c0]

  It successively moves a 1 from the source into each of the flag bits in the list shown above.

SRC	Destination	Message Reference
`zflag`	Each flag bit	#1–#20
`nflag`	Each flag bit	#21–#40
`tflag`	Each flag bit	#41–#60

SRC	Destination	Message Reference
fflag	Each flag bit	#61–#80
rflag	Each flag bit	#81–#100
64[c0]	Each flag bit	#101–#120

After each move, it checks that lflag is set properly. After moving into all of the flag bits, it checks that each was set properly.

- MoveTests4c: MOV1 Clear Flag Bits Test (part 1) — This is a test of the MOV1 instruction. It uses the following bits as operands:

  lsb
  lflag
  cflag
  vflag
  zflag
  nflag
  tflag
  fflag
  rflag
  64[c0]

  It successively moves a 0 from the source into each of the flag bits in the list shown above.

SRC	Destination	Message Reference
lsb	Each flag bit	#1–#20
lflag	Each flag bit	#21–#40
cflag	Each flag bit	#41–#60
vflag	Each flag bit	#61–#80
zflag	Each flag bit	#81–#100
nflag	Each flag bit	#101–#120

After each move, it checks that lflag is cleared properly. After moving into all of the flag bits, it checks that each was cleared properly.

The remainder of this test is continued in MoveTests4d.ma.

- MoveTests4d: MOV1 Clear Flag Bits Test (part 2) — This is a test of the MOV1 instruction. It uses the following bits as operands:

  lsb
  lflag
  cflag
  vflag
  zflag
  nflag
  tflag
  fflag
  rflag
  64[c0]

It successively moves a 0 from the source into each of the flag bits shown above.

SRC	Destination	Message Reference
tflag	Each flag bit	#1–#20
fflag	Each flag bit	#21–#40
rflag	Each flag bit	#41–#60
64[c0]	Each flag bit	#61–#80

After each move, it checks that lflag is cleared properly. After moving into all of the flag bits, it checks that each was cleared properly.

- `MoveTests4e`: MOV1 MOVE PReg To Flag-Bit Test — This test uses the following bits as operands:

Flags	PRegs
lsb	1026[c2] — c2 = 0
lflag	1027[c2] — c4 = 1024
cflag	1028[c2]
vflag	1029[c2]
zflag	1030[c2]
nflag	1031[c2]
tflag	1032[c2]
fflag	1033[c2]
rflag	1034[c2]
2[c4]	1035[c2]
	1036[c2]

The test starts by testing the ability to perform the MOV1 function between the following sources and destinations:

SRC	Destination	Message Reference
1025[c2]	Each flag bit	#1–#22
Each flag bit	Each PReg	#23–#44

If it detects an error in the result, it prints an error message giving the instruction under test, the source and destination operands, the expected result and the actual result.

- `MoveTests4f`: MOV1 Immediate To PReg Test — Enables all PEs, then uses MOV1 to move a 1 into eight different PReg locations.

  Repeats the above operation moving a 0 into the eight PReg locations.

  If the test detects an error, it prints an error message giving the operation attempted, the expected result and the actual result.

- `MoveTests5a`: MOV1UC—Flag To Flag 1's Test (part 1) — This is a test of the ability of the MOV1UC instruction to move a 1 from flag to flag. It uses the following bits as operands:

  lsb

lflag
cflag
vflag
zflag
nflag
tflag
fflag
rflag
64[c0]

First, it tests that the MOV1UC instruction can move an immediate 1 or 0 into each of the above flag bits, and that the lflag is correct after each move (message #1–#41).

Next, it successively moves a 1 from the source into each of the flag bits in the list shown above.

SRC	Destination	Message Reference
lsb	Each flag bit	#42–#51
lflag	Each flag bit	#52–#61
cflag	Each flag bit	#62–#71
vflag	Each flag bit	#72–#81

After moving 1 into all of the flag bits, it checks that each was set properly.

If the test detects an error, it prints an error message giving the operation, the expected result and the actual result.

This test is continued in MoveTests5b.ma

- `MoveTests5b`: MOV1UC Flag To Flag 1's Test (part 2) — This is a test of the ability of the MOV1UC instruction to move a 1 from flag to flag. It uses the following bits as operands:

  lsb
  lflag
  cflag
  vflag
  zflag
  nflag
  tflag
  fflag
  rflag
  64[c0]

  It successively moves a 1 from the source into each of the flag bits in the list shown above.

SRC	Destination	Message Reference
zflag	Each flag bit	#1–#10
nflag	Each flag bit	#11–#20
tflag	Each flag bit	#21–#30
fflag	Each flag bit	#31–#40

SRC	Destination	Message Reference
rflag	Each flag bit	#41–#50
64[c0]	Each flag bit	#51–#60

After moving 1 into all of the flag bits, it checks that each was set properly.

If the test detects an error, it prints an error message giving the operation, the expected result and the actual result.

- `MoveTests5c`: MOV1UC Flag To Flag 0's Test (Part 1) — This is a test of the ability of the MOV1UC instruction to move a 0 from flag to flag. It uses the following bits as operands:

  lsb
  lflag
  cflag
  vflag
  zflag
  nflag
  tflag
  fflag
  rflag
  64[c0]

  It successively moves a 0 from the source into each of the flag bits in the list shown above.

SRC	Destination	Message Reference
~lsbag	Each flag bit	`#1--#10`
~lflag	Each flag bit	`#11--#20`
~cflag	Each flag bit	`#21--#30`
~vflag	Each flag bit	`#31--#40`
~zflag	Each flag bit	`#41--#50`
~nflag	Each flag bit	`#51--#60`

  After moving 0 into all of the flag bits, it checks that each was cleared properly.

  If the test detects an error, it prints an error message giving the operation, the expected result and the actual result.

  This test is continued in MoveTests5d.ma

- `MoveTests5d`: MOV1UC Flag To Flag 0's Test (part 2) — This is a test of the ability of the 'mov1uc' instruction to move a 0 from flag to flag.

  It uses the following bits as operands:

  lsb
  lflag
  cflag
  vflag
  zflag
  nflag
  tflag
  fflag

rflag
64[c0]

It successively moves a 0 from the source into each of the flag bits in the list shown above.

SRC	Destination	Message Reference
~tsfag	Each flag bit	#1–#10
~fflag	Each flag bit	#11–#20
~rflag	Each flag bit	#21–#30
~64[c0]	Each flag bit	#31–#40

After moving 0 into all of the flag bits, it checks that each was cleared properly.

If the test detects an error, it prints an error message giving the operation, the expected result and the actual result.

- `MoveTests5f`: MOV1UC Immediate To PReg Test — Moves a 1 into 8 different PReg locations, then checks for errors; moves a 0 into the same PReg locations, then checks for errors.

  If the test detects an error, it prints an error message giving the attempted operation, the expected results and the actual results.

- `MoveTests6`: 8-Bit & 32-Bit MOVE Test — Performs the following tests:

  - Test #1 — Makes the following 32-bit moves and checks the result:

    `0x5e461237 --> 16[c0] --> 1040[c0]`

  - Test #2 — Makes the following 8-bit moves and checks the result:

    `0xa5 --> 0[c0] --> 1024[c0]`

  - Test #3 — Makes the following 32-bit moves and checks the result:

    `0x12345678 --> acc --> 1040[c0]`

  - Test #4 — Makes the following 8-bit moves and checks the result:

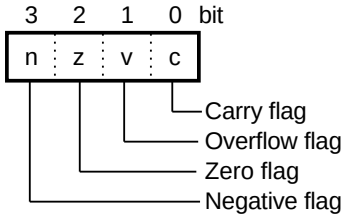
    `0xb6 --> acc --> 1024[c0]`

  If the test detects an error, it prints an error message giving the attempted operation, the expected results and the actual results.

- `OrTests1`: Register To Register OR Test — This test performs 8-bit, 16-bit, 32-bit and 64-bit OR operations.

  After each OR, if it detects an error, the test prints an error message which gives the OR the test was attempting, the expected answer and the actual answer.

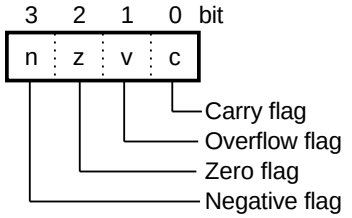
  After each OR, it checks the flag bits. If any of these is in error, an error message is printed giving the operation attempted, the expected flag bits and the actual flag bits. The flag bits are collected as shown:



- `OrTests2`: Immediate To Register OR Test — This test performs 8-bit, 16-bit, 32-bit and 64-bit OR operations.

  After each OR, if it detects an error, the test prints an error message which gives the OR the test was attempting, the expected answer and the actual answer.

  After each OR, it checks the flag bits. If any of these is in error, an error message is printed giving the operation attempted, the expected flag bits and the actual flag bits. The flag bits are collected as shown:



- `ShiftTests`: Shift Instruction Test — Tests the four shift instructions:
  - shll: Logical shift left
  - shla: Arithmetic shift left
  - shrl: Logical shift right
  - shra: Arithmetic shift right

  It performs 4-bit and 7-bit shifts, using a variety of operands:
  - Test #1 (message #1–#40): This test uses the value c1 = 4 to determine how many places to shift the contents of 16[c0].
  - Test #2 (message #41–#80): This test uses the value c1 = 7 to determine how many places to shift the contents of 16[c0].
  - Test #3 (message #81–#120): This test uses an immediate number in the instruction to determine how many places to shift the contents of 16[c0].
  - Test #4 (message #121–#160): This test uses an immediate number in the instruction to determine how many places to shift the contents of acc.
  - Test #5 (message #161–#180): This test uses the value 128[c0] = 4 to determine how many places to shift the contents of 16[c0].
  - Test #6 (message #181–#200): This test uses the value acc = 4 to determine how many places to shift the contents of 16[c0].

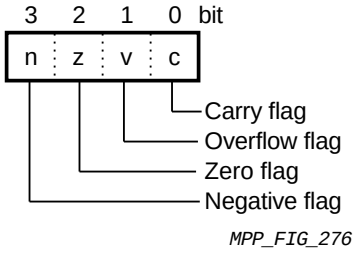
- Test #7 (message #201–#240): This test uses the value in 128[c0] to determine how many places to shift the contents of acc.
- Test #8 (message #241–#260): This test uses the value 128[c0] = 7 to determine how many places to shift the contents of 16[c0].
- Test #9 (message #261–#280): This test uses the contents of acc to determine how many places to shift the contents of 16[c0].
- Test #10 (message #281–#300): This test uses the contents of 128[c0] to determine how many places to shift the contents of acc.
- Test #11 (message #301–#320): This test uses the contents of acc to determine how many places to shift. It also shifts the contents of acc.

If the test detects an error, it prints an error message giving the attempted operation, the expected results and the actual results.

- SubTests1: Register To Register SUBTRACTION Test — This test enables all PEs and has them load data into PReg. Then it performs various 8-bit, 16-bit, 32-bit and 64-bit subtractions between PRegs.

  After each SUBTRACTION, if it detects an error, the test prints an error message which gives the SUBTRACTION the test was attempting, the expected answer and the actual answer.

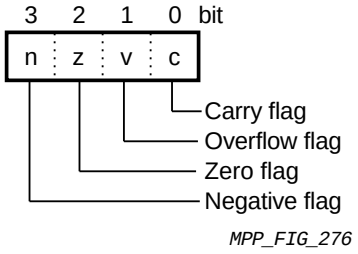
  After each SUBTRACTION, it checks the flag bits. If any of these is in error, the test prints an error message giving the operation attempted, the expected flag bits and the actual flag bits. The flag bits are collected as shown:



- SubTests2: Immediate To Register SUBTRACTION Test — This test enables all PEs, then performs various 8-bit, 16-bit, 32-bit and 64-bit subtractions between immediate and PRegs.

  After each SUBTRACTION, if it detects an error, the test prints an error message which gives the SUBTRACTION the test was attempting, the expected answer and the actual answer.

  After each SUBTRACTION, it checks the flag bits. If any of these is in error, the test prints an error message giving the operation attempted, the expected flag bits and the actual flag bits. The flag bits are collected as shown:



- `div1`: 64-Bit Signed Integer DIVIDE Test — This test enables all the PEs, then performs a series of 64-bit divisions. After each division, the test inverts all the bits of the answer and saves this separately.

  If an error occurs, the test prints an error message which gives the attempted division, the expected result, the actual result and whether or not this is the bit-inverted version of the answer. Typically, if the answer is wrong, the bit inversion is also wrong, and the test prints a separate message for each.

- `divu0`: 64-Bit Unsigned Integer DIVIDE Test — This test enables all the PEs, then performs a series of 64-bit unsigned divisions.

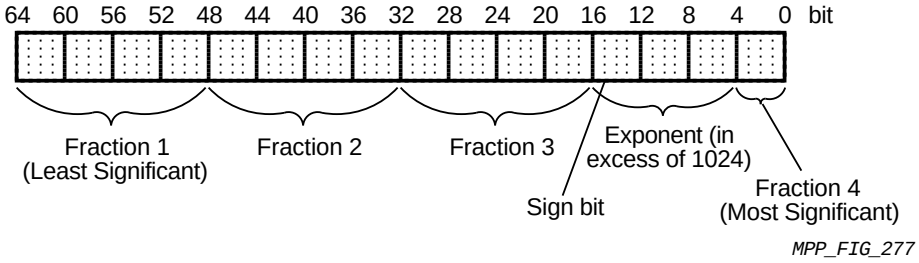
  If an error occurs, the test prints an error message which gives the attempted division, the expected result and the actual result.

- `ebit1`: E-Bit Operation Test — This test verifies that when the e-bit is turned on, data transfers to PReg can take place, and that when the e-bit is turned off, data transfers to PReg are blocked.

  If it detects an error, this test prints an error message which gives the attempted operation, the expected result and the actual result.

- `fadd1`: 64-Bit Floating Point ADD Test — This tests performs two simple 64-bit floating point additions: `1.0 + 2.0 = ?` and `2.0 + 1.0 = ?`.

  If it detects an error, this test prints an error message which gives the attempted operation, the expected result and the actual result. Both the expected and the actual result is expressed in VAX floating point g-format. This is a 64-bit integer whose fields have the following meaning:

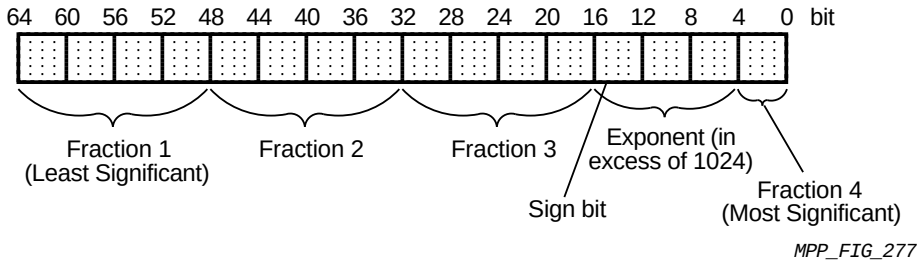


- `fmult1`: 64-Bit Floating Point MULTIPLY Test — This test performs a series of 64-bit floating point multiplication operations.

  It enables all the PEs, then performs a series of 64-bit multiplications. After each operation, the test inverts all the bits of the answer and saves this separately.

  If an error occurs, the test prints an error message which gives the attempted operation, the expected result, the actual result and whether or not this is the bit-inverted version of the answer. Typically, if the answer is wrong, the bit inversion is also wrong, and the test prints a separate message for each.

  Both the expected and the actual result is expressed in VAX floating point g-format. This is a 64-bit integer whose fields have the following meaning:



- `msol0`: Brief Solitary LDSOL64/STSOL64 Test — This test enables a single PE per cluster for load/store operations. It performs a solitary 64-bit store and several solitary 64-bit load operations, and verifies that the data transfers correctly. It then verifies that none of the data reached the unselected PEs.

  If an error occurs, the test prints an error message which gives the attempted operation, the expected result and the actual result.

- `msol1`: Indirectly Addressed LDSOL/STSOL Test — Performs the following tests:

  - Test #1 — With a single PE per cluster enabled for load/store operation, the test uses the solitary load/store commands to move data to and from PMem, using indirect addressing. It verifies that the data has transferred correctly, and that no data has transferred to disabled PEs. (message #1–#10)
  - Test #2 — Repeats test #1, using different data and initialization patterns. (message #11–#20)
  - Test #3 — Uses the regular load command to read the data stored by the solitary store command in the previous test, and to verify that no data has transferred to disabled PEs (message #21–#28). In this test, it enables all PEs for load/store before loading the data into PReg. Then it enables only the single PE per cluster to verify the data in the PReg.

  The following table summarizes the test operations. Notice the different ways test #2 and test #3 use to verify that the PMem of unselected PEs remained as originally initialized.

Test	Single PE	All Others	Operation	Result
Init:	mbit = 1	mbit = 1	init –> PMem	
Test 1-2:	mbit = 1	mbit = 0	data –> PMem –> PReg	Data OK?
	mbit = 0	mbit = 1	PMem –> PReg	Unchanged?
Test 3:	mbit = 1	mbit = 1	PMem –> PReg	
	ebit = 1	ebit = 0	PReg	Data OK?
	ebit = 0	ebit = 1	PReg	Unchanged?

  If an error occurs, the test prints an error message which gives the attempted operation, the expected result and the actual result.

- `msol2`: Indirectly Addressed Single PE LOAD/STORE Test — This test is identical to `msol1.ma`, except that it uses the regular LD/ST commands instead of the specialized LDSOL/STSOL commands.

- `mtest0`: LOAD/STORE Overlap Test — The purpose of this test is to verify the ability to perform overlapping operations by overlapping store, multiplication and load operations. It is possible for the multiplication process to corrupt the load and store data; but it is much more likely that the process of loading and storing data upsets the multiplication process.

  This test sets the e-bit and m-bit on all PEs. It then performs the following operations in succession:

  - Each PE stores test data into 8 different PMem locations
  - Each PE performs five 64-bit multiplications
  - Each PE loads test data from the 8 different PMem locations
  - Checks test data
  - Checks multiplication data

  If an error occurs, the test prints an error message which gives the attempted operation, the expected result and the actual result.

- `mtest1`: Dispatcher Lock And M-Bit Queue Test — Performs these tests:

  - Test #1 (Message #1–#2) — Tests the m-bit queue and dispatcher lock, and verifies that the m-bits are correctly set and cleared.

    First, it uses MOV1UC to clear all m-bits and verifies that they are all cleared. In rapid succession, it issues 32 commands to clear the m-bits, and then a command to set the m-bits. It checks that the m-bits are set. If there is a fault in the queue mechanism or dispatcher lock, the command to set the m-bit may be lost.

  - Test #2 (Message #3–#5) — Tests the action of the m-bits. It verifies that turning on the m-bits allows a store, that turning off the m-bits prevents a store and that turning off the m-bits prevents a load.

  - Test #3 (Message #6) — Dispatcher lock test; tests to see if, with the m-bit queue full, the dispatcher lock prevents subsequent commands from overflowing the queue. It verifies not merely that the m-bits are correctly set or cleared, but that they are functional at the proper time.

    With all m-bits on, the test stores and loads data. In rapid succession it issues 32 commands to clear the m-bits (filling up the queue), then sets the m-bits and attempts to load the data originally stored. It verifies that it receives the correct data.

  If an error occurs, the test prints an error message which describes the suspected failure.

- `mtest2`: Indirect LOAD/STORE Tests — This test verifies the ability of the PEs to store and load indirectly addressed data to and from 0x1000 locations in PMem.

  The test enables all the PEs for load/store operations, and gives each PE unique data to store and load. Using indirect addressing, it stores data at 0x1000 PMem locations, incrementing the data for each location. Still using indirect addressing, it reads back the data from all those PMem locations and verifies the accuracy of the data. Finally, using direct addressing, it reads back the data from all those PMem locations and again verifies its accuracy.

  Finally, the test has all PEs invert the bits in their original data and repeat the above test.

- `mtest3`: Simple Indirectly Addressed LOAD/STORE Test — Performs a 64-bit, 32-bit, 16-bit and 8-bit indirectly addressed store followed by a similar set of indirectly addressed loads from the same locations. Checks the accuracy of the data. Repeats the above using different data.

  If an error occurs, the test prints an error message which gives the attempted operation, the expected result and the actual result.

- `mtest4`: 64-Bit Directly Addressed LOAD/STORE Test — Performs 64-bit stores to PMem starting with 0(c0) and ending with 16320(c0). After each store, performs a 64-bit load from the same location and checks the result.

  If an error occurs, the test prints an error message which gives the attempted operation, the expected result and the actual result.

- `mtest5`: 32-Bit Byte Offset LOAD/STORE Test — At 0-byte offset from longword boundary, this test stores (writes) and loads (reads) data from PMem. Each PE uses its own row/column as data which it increments for each of 0xfff 32-bit locations. It checks that the same data is read back from each location. It repeats the test using inverted data.

  It then offsets the PMem addresses by a single byte and repeats for an offset of 1, 2, and 3 bytes from a longword boundary. After doing this for each of the 4 possible byte offsets, it repeats the process 128 times.

  If an error occurs, the test prints an error message which gives the attempted operation, the expected result and the actual result.

- `mult1`: MULTIPLY Test — This test performs the following multiplication operations, between the source and destination as shown below. It checks the result after each operation.

Operation	SRC	Dest	Error Message
mul64:	PReg	PReg	#1–#8
mul32:	PReg	PReg	#9–#14
mul32:	Immediate	PReg	#15–#16
mul32:	Neg immediate	PReg	#17–#18
mul32:	Immediate	PReg	#19–#20
mul32:	Neg immediate	PReg	#21–#22
mulu32:	Immediate	PReg	#23–#24
mul8:	CReg	PReg	#25–#26
mul16:	CReg	PReg	#27–#28
mul32:	CReg	PReg	#29–#30
mul64:	CReg	PReg	#31–#34
mul64:	Immediate	PReg	#35–#38

  If an error occurs, the test prints an error message which gives the attempted operation, the expected result and the actual result.

- `onet`: XNet Octagon Shift Test — Tests the ability of the PEs to shift data around an octagon: north, northeast, east, southeast, south, southwest, west and northwest. Each PE uses its own row, column address as data and shifts it in each of the eight directions. After the eight shifts, each PE does an

exclusive OR of the shifted result with the starting data. If no corruption occurred, the result is zero.

The test begins with a distance of zero (shifts to itself), and successively widens the distance to a distance of 2047.

If an error occurs with any PE, the test prints an error message which gives the distance shifted, the expected state of the error status bit and the actual state of the error status bit (which, of course, is 1 or the message would not have been printed in the first place).

- `onet1`: XNet Octagon Shift Test (with PMem activity) — Tests the ability of the PEs to shift data around an octagon: north, northeast, east, southeast, south, southwest, west and northwest. Each PE uses its own row, column address as data and shifts it in each of the eight directions. After each shift, the test performs several store operations to create extra activity in the m-machine.

  After the eight shifts, each PE does an exclusive OR of the shifted result with the starting data. If no corruption occurred, the result is zero.

  The test begins with a distance of zero (shifts to itself), and successively widens the distance to a distance of 2047.

  If an error occurs with any PE, the test prints an error message which gives the distance shifted, the expected state of the error status bit and the actual state of the error status bit (which, of course, is 1 or the message would not have been printed in the first place).

- `ortest1`: Global OR And XOR Test — With all PEs enabled, each PE moves data into PReg. Then all PEs perform a global OR of the PRegs into an ACU register, the contents of which is checked for error.

  Next, each PE uses the XOR function to invert all the bits of the data in PReg. A global OR operation is then performed and result is checked for error.

  If an error occurs, the test prints an error message giving the operation attempted, the PReg in question, the expected result and the actual result.

- `random1`: Random Macro Instruction Test — This test performs instructions of all sizes and many types in random sequence. Occasionally, it checks the contents of the CPSW status register for unexpected status. After each instruction tested, it checks the results for error.

  If an error occurs, it prints an error message giving the instruction under test, the expected result and the actual result.

- `rt0a`: 16-Bit Router Send Test — Sets the e-bit and the t-bit for PE 0,0 only. Opens the router channel (to itself) and sends data using the router. It checks the accuracy of the data and also checks that adjacent data remains unchanged. This demonstrates not only that the data gets sent, but also that only 16 bits are sent.

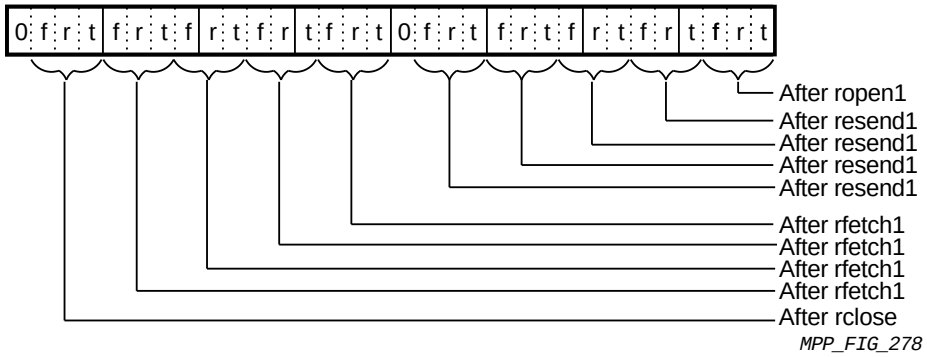
- `rt0b`: 16-Bit Router Fetch Test — Sets the e-bit and the t-bit for PE 0,0 only. Opens the router channel (to itself) and fetches data using the router. It checks the accuracy of the data and also checks that adjacent data remains unchanged. This demonstrates not only that the data gets fetched, but also that only 16 bits are fetched.

- `rt0c`: Router Send Test (1, 8, 16, 32, 64 Bits) — Sets the e-bit and the t-bit for PE 0,0 only. Opens the router channel (to itself) and sends data using the router. It performs 1-bit, 8-bit, 16-bit, 32-bit and 64-bit send operations. This test demonstrates not only that the data gets sent, but also that only the specified number of bits are sent.
- `rt0d`: Router Fetch Test (1, 8, 16, 32, 64 Bits) — Sets the e-bit and the t-bit for PE 0,0 only. Opens the router channel (to itself) and fetches data using the router. It performs 1-bit, 8-bit, 16-bit, 32-bit and 64-bit fetch operations. This test demonstrates not only that the data gets fetched, but also that only the specified number of bits are fetched.
- `rt0e`: Router Send And Fetch Test — This test performs router send and fetch operations. It disables all PEs except #0. The source and destination is PE #0. It verifies that the correct data reaches the destination and that the correct tflag, rflag and fflag is set after each operation. It also verifies that no data reaches disabled PEs, and none of their flags are affected.

  At each stage of the test, data and flags are checked. If an error occurs, the test prints an error message.

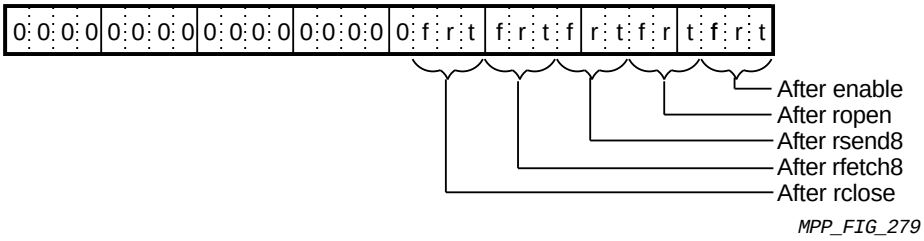
  This is the meaning of the expected and actual results mentioned in the following error messages:

  - Message #1: The test attempts to send 0xa one bit at a time with four `rsend1` commands. The lower order word of the result was not as expected.
  - Message #2: The high order word of the result was not as expected.
  - Message #3: The test attempts to fetch 0xb one bit at a time with four `rfetch1` commands. The lower order word of the result was not as expected.
  - Message #4: The high order word of the result was not as expected.
  - Message #5: Each cluster of 3 bits shown below represents the state of the fflag, the rflag and the tflag after each stage of the test. Please note that bit 15 and bit 31 are not used.



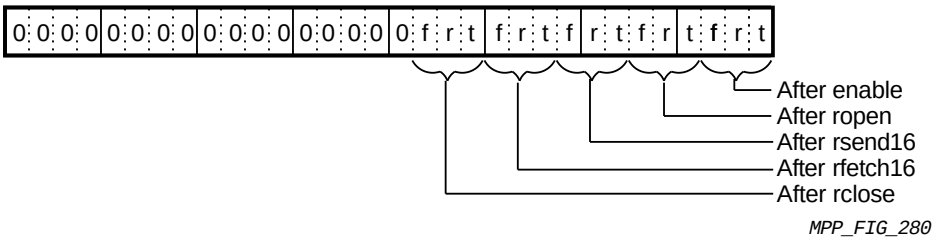
  - Messages #6–9: These words are comparable to words #1-4, except they are from the unselected PEs. Normally, no data reaches these destinations, and they remain at zero.
  - Message #10: This displays the flag bits from the unselected PEs which should all remain at zero.

- Message #11: The test attempts to send 0x8a using the `rsend8` command. This word contains the low order bits of that operation.
- Message #12: This is the higher order bits of the destination. All these bits should remain zero.
- Message #13: The test attempts to fetch 0x8b using the `rfetch8` command. This word contains the low order bits of that operation.
- Message #14: This is the higher order bits of the destination. All these bits should remain zero.
- Message #15: Each cluster of 3 bits represents the state of the fflag, the rflag and the tflag after each stage of the test:



MPP_FIG_279

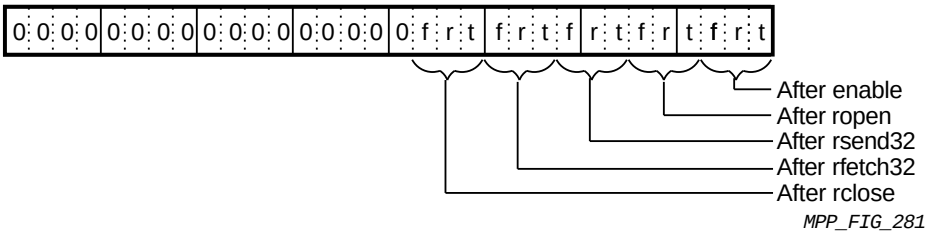
- Messages #16–19: These words are comparable to words #11–14, except they are from the unselected PEs. Normally, no data reaches these destinations, and they remain at zero.
- Message #20: This displays the flag bits from the unselected PEs which should all remain at zero.
- Message #21: The test attempts to send 0x7a8a using the `rsend16` command. This word contains the low order bits of that operation.
- Message #22: This is the higher order bits of the destination. All these bits should remain zero.
- Message #23: The test attempts to fetch 0x7b8b using the `rfetch16` command. This word contains the low order bits of that operation.
- Message #24: This is the higher order bits of the destination. All these bits should remain zero.
- Message #25: Each cluster of 3 bits represents the state of the fflag, the rflag and the tflag after each stage of the test:



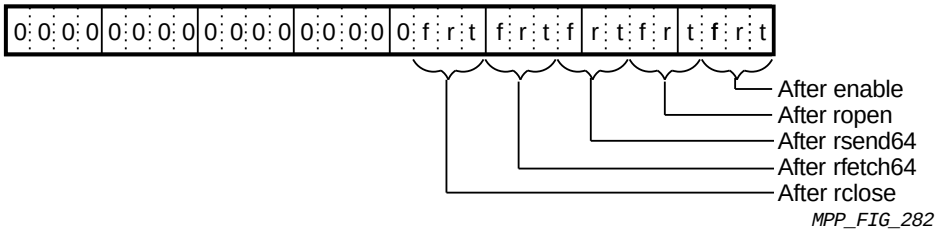
MPP_FIG_280

- Messages #26–29: These words are comparable to words #21–24, except they are from the unselected PEs. Normally, no data reaches these destinations, and they remain at zero.

- Message #30: This displays the flag bits from the unselected PEs which should all remain at zero.
- Message #31: The test attempts to send 0x5a6a7a8a using the `rsend32` command. This word contains the result of that operation.
- Message #32: This is the higher order bits of the destination. All these bits should remain zero.
- Message #33: The test attempts to fetch 0x5b6b7b8b using the `rfetch32` command. This word contains the result of that operation. The higher order bits should remain zero.
- Message #34: This is the higher order bits of the destination. All these bits should remain zero.
- Message #35: Each cluster of 3 bits represents the state of the fflag, the rflag and the tflag after each stage of the test.



- Messages #36–39: These words are comparable to words #31–34, except they are from the unselected PEs. Normally, no data reaches these destinations, and they remain at zero.
- Message #40: This displays the flag bits from the unselected PEs which should all remain at zero.
- Message #41: The test attempts to send 0x5a6a7a8a using the `rsend64` command. This word contains the low order bits of that operation.
- Message #42: This is the higher order bits of the destination.
- Message #43: The test attempts to fetch 0x5b6b7b8b using the `rfetch64` command. This word contains the low order bits of that operation.
- Message #44: This is the higher order bits of the destination.
- Message #45: Each cluster of 3 bits represents the state of the fflag, the rflag and the tflag after each stage of the test.



- Messages #46–49: These words are comparable to words #41-44, except they are from the unselected PEs. Normally, no data reaches these destinations, and they remain at zero.

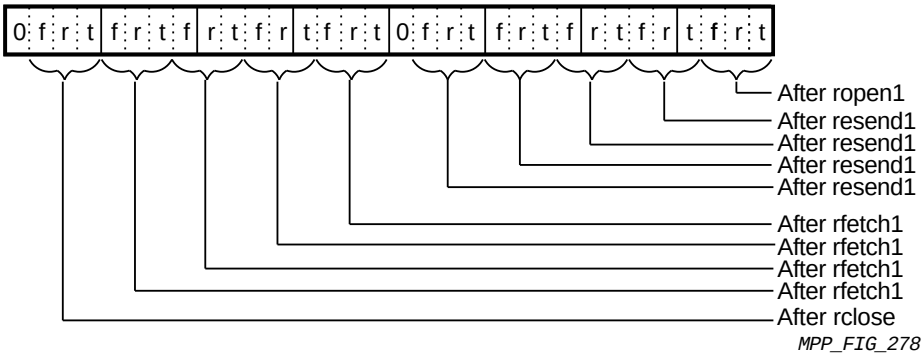
- Message #50: This displays the flag bits from the unselected PEs which should all remain at zero.

• `rt0f`: Router Send And Fetch (While Storing) Test — This test performs router send and fetch operations while the m-machine is busy doing 64-bit store operations. It disables all PEs except #0. The source and destination is PE #0. It verifies that the correct data reaches the destination and that the correct tflag, rflag and fflag is set after each operation. It also verifies that no data reaches disabled PEs, and none of their flags are affected.

  At each stage of the test, data and flags are checked. If an error occurs, the test prints an error message.

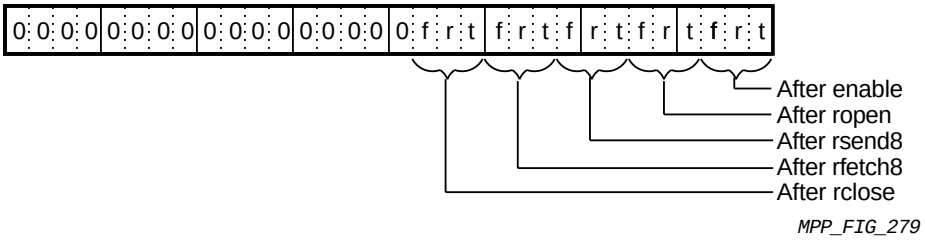
  This is the meaning of the expected and actual results mentioned in the following error messages:

  - Message #1: The test attempts to send 0xa one bit at a time with four `rsend1` commands. The lower order word of the result was not as expected.
  - Message #2: The high order word of the result was not as expected.
  - Message #3: The test attempts to fetch 0xb one bit at a time with four `rfetch1` commands. The lower order word of the result was not as expected.
  - Message #4: The high order word of the result was not as expected.
  - Message #5: Each cluster of 3 bits represents the state of the fflag, the rflag and the tflag after each stage of the test. Please note that bit 15 and bit 31 are not used.



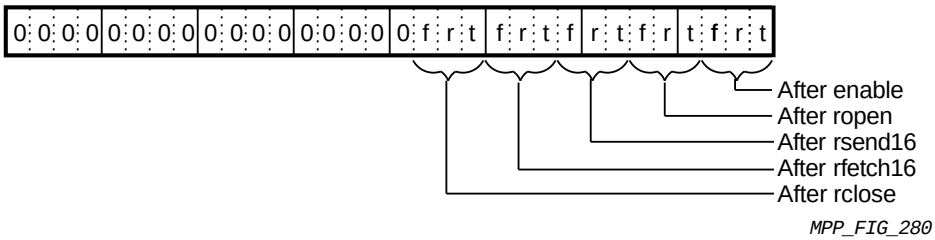
  - Messages #6–9: These words are comparable to words #1–4, except they are from the unselected PEs. Normally, no data reaches these destinations, and they remain at zero.
  - Message #10: This displays the flag bits from the unselected PEs which should all remain at zero.
  - Message #11: The test attempts to send 0x8a using the `rsend8` command. This word contains the low order bits of that operation.
  - Message #12: This is the higher order bits of the destination. All these bits should remain zero.
  - Message #13: The test attempts to fetch 0x8b using the `rfetch8` command. This word contains the low order bits of that operation.

- Message #14: This is the higher order bits of the destination. All these bits should remain zero.
- Message #15: Each cluster of 3 bits represents the state of the fflag, the rflag and the tflag after each stage of the test.



MPP_FIG_279

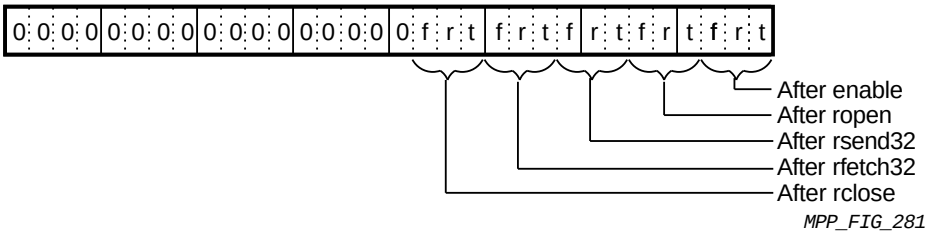
- Messages #16–19: These words are comparable to words #11–14, except they are from the unselected PEs. Normally, no data reaches these destinations, and they remain at zero.
- Message #20: This displays the flag bits from the unselected PEs which should all remain at zero.
- Message #21: The test attempts to send 0x7a8a using the `rsend16` command. This word contains the low order bits of that operation.
- Message #22: This is the higher order bits of the destination. All these bits should remain zero.
- Message #23: The test attempts to fetch 0x7b8b using the `rfetch16` command. This word contains the low order bits of that operation.
- Message #24: This is the higher order bits of the destination. All these bits should remain zero.
- Message #25: Each cluster of 3 bits represents the state of the fflag, the rflag and the tflag after each stage of the test.



MPP_FIG_280

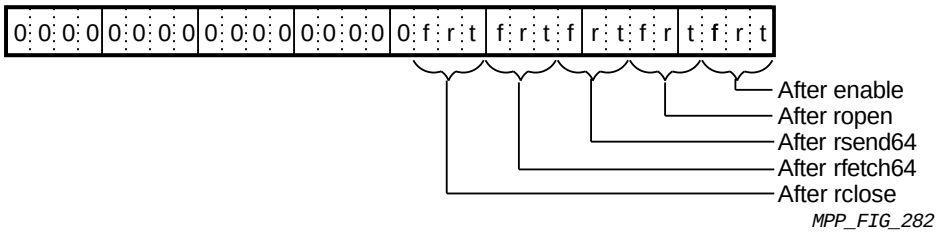
- Messages #26–29: These words are comparable to words #21–24, except they are from the unselected PEs. Normally, no data reaches these destinations, and they remain at zero.
- Message #30: This displays the flag bits from the unselected PEs which should all remain at zero.
- Message #31: The test attempts to send 0x5a6a7a8a using the `rsend32` command. This word contains the result of that operation.
- Message #32: This is the higher order bits of the destination. All these bits should remain zero.

- Message #33: The test attempts to fetch 0x5b6b7b8b using the `rfetch32` command. This word contains the result of that operation. The higher order bits should remain zero.
- Message #34: This is the higher order bits of the destination. All these bits should remain zero.
- Message #35: Each cluster of 3 bits represents the state of the fflag, the rflag and the tflag after each stage of the test.



MPP_FIG_281

- Messages #36–39: These words are comparable to words #31–34, except they are from the unselected PEs. Normally, no data reaches these destinations, and they remain at zero.
- Message #40: This displays the flag bits from the unselected PEs which should all remain at zero.
- Message #41: The test attempts to send 0x5a6a7a8a using the `rsend64` command. This word contains the low order bits of that operation.
- Message #42: This is the higher order bits of the destination.
- Message #43: The test attempts to fetch 0x5b6b7b8b using the `rfetch64` command. This word contains the low order bits of that operation.
- Message #44: This is the higher order bits of the destination.
- Message #45: Each cluster of 3 bits represents the state of the fflag, the rflag and the tflag after each stage of the test.



MPP_FIG_282

- Messages #46–49: These words are comparable to words #41–44, except they are from the unselected PEs. Normally, no data reaches these destinations, and they remain at zero.
- Message #50: This displays the flag bits from the unselected PEs which should all remain at zero.

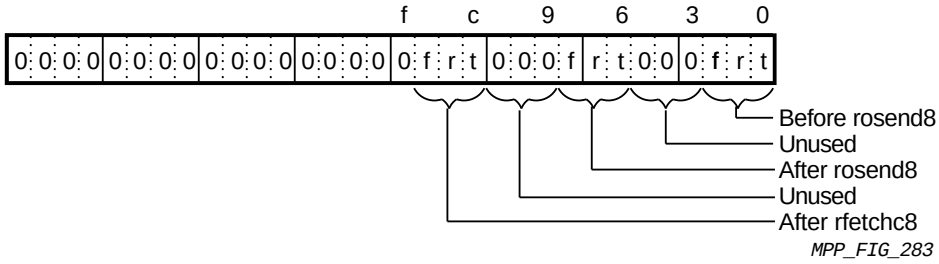
- `rt0g`: Router Open/Send And Fetch/Close Test — This test performs router open/send and fetch operations. It disables all PEs except #0. The source and destination is PE #0. It verifies that the correct data reaches the destination and that the correct tflag, rflag and fflag is set after each operation. It

also verifies that no data reaches disabled PEs, and none of their flags are affected.

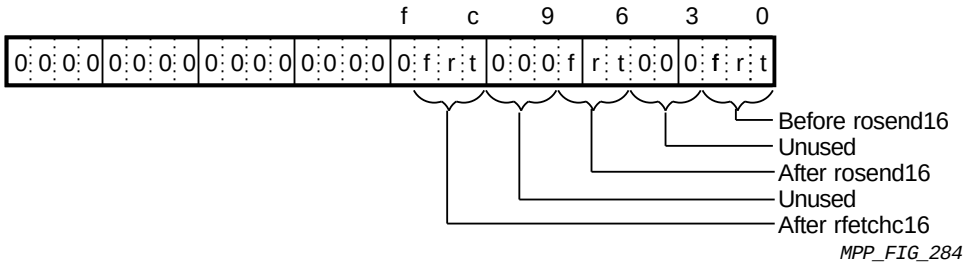
At each stage of the test, data and flags are checked. If an error occurs, the test prints an error message.

This is the meaning of the expected and actual results mentioned in the following error messages:

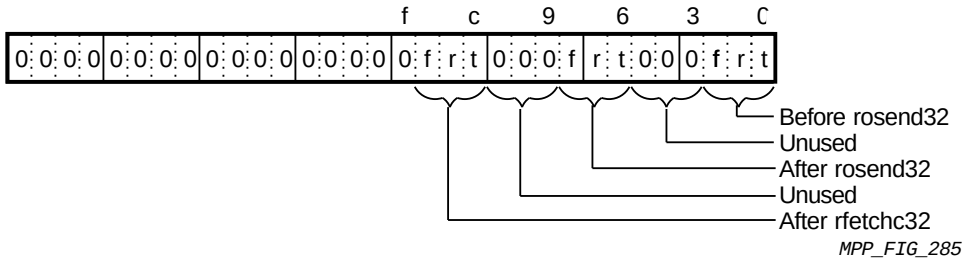
- Message #1: The test attempts to open/send 0x8a using the `rosend8` command. This word contains the low order bits of that operation.
- Message #2: This is the higher order bits of the destination. All these bits should remain zero.
- Message #3: The test attempts to fetch/close 0x8b using the `rfetchc8` command. This word contains the low order bits of that operation.
- Message #4: This is the higher order bits of the destination. All these bits should remain zero.
- Message #5: Each cluster of 3 bits represents the state of the fflag, the rflag and the tflag after each stage of the test.



- Messages #6–9: These words are comparable to words #1–4, except they are from the unselected PEs. Normally, no data reaches these destinations, and they remain at zero.
- Message #10: This displays the flag bits from the unselected PEs which should all remain at zero.
- Message #11: The test attempts to send 0x7a8a using the `rsend16` command. This word contains the low order bits of that operation.
- Message #12: This is the higher order bits of the destination. All these bits should remain zero.
- Message #13: The test attempts to fetch 0x7b8b using the `rfetch16` command. This word contains the low order bits of that operation.
- Message #14: This is the higher order bits of the destination. All these bits should remain zero.
- Message #15: Each cluster of 3 bits represents the state of the fflag, the rflag and the tflag after each stage of the test.

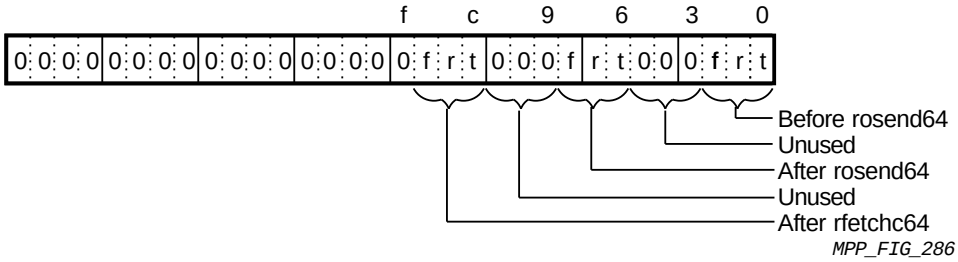


- Messages #16–19: These words are comparable to words #11–14, except they are from the unselected PEs. Normally, no data reaches these destinations, and they remain at zero.
- Message #20: This displays the flag bits from the unselected PEs which should all remain at zero.
- Message #21: The test attempts to send 0x5a6a7a8a using the `rsend32` command. This word contains the low order bits of that operation.
- Message #22: This is the higher order bits of the destination. All these bits should remain zero.
- Message #23: The test attempts to fetch 0x5b6b7b8b using the `rfetch32` command. This word contains the low order bits of that operation.
- Message #24: This is the higher order bits of the destination. All these bits should remain zero.
- Message #25: Each cluster of 3 bits represents the state of the fflag, the rflag and the tflag after each stage of the test.



- Messages #26–29: These words are comparable to words #21–24, except they are from the unselected PEs. Normally, no data reaches these destinations, and they remain at zero.
- Message #30: This displays the flag bits from the unselected PEs which should all remain at zero.
- Message #31: The test attempts to send 0x5a6a7a8a using the `rsend64` command. This word contains the low order bits of that operation.
- Message #32: This is the higher order bits of the destination.
- Message #33: The test attempts to fetch 0x5b6b7b8b using the `rfetch64` command. This word contains the low order bits of that operation.
- Message #34: This is the higher order bits of the destination.

- Message #35: Each cluster of 3 bits represents the state of the fflag, the rflag and the tflag after each stage of the test.



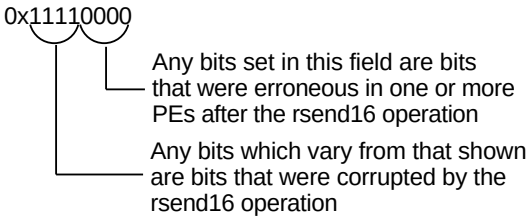
MPP_FIG_286

- Messages #36–39: These words are comparable to words #31–34, except they are from the unselected PEs. Normally, no data reaches these destinations, and they remain at zero.
- Message #40: This displays the flag bits from the unselected PEs which should all remain at zero.

- `rt1a`: Single PE Per Cluster Router Send And Fetch Test — This test enables one PE per cluster for router operation. It does a 16-bit router send and a 16-bit router fetch operation. Each PE has unique send data and unique fetch data based on the PEs address. The state of the tflag, rflag, and fflag is saved at each step in the test. When finished, the test checks for error and prints an error message if any errors are found.

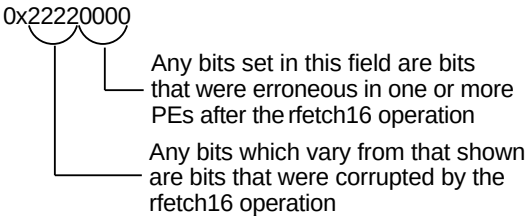
  Following is an explanation of the expected and actual result printed in the error message:

  - Error message #1:



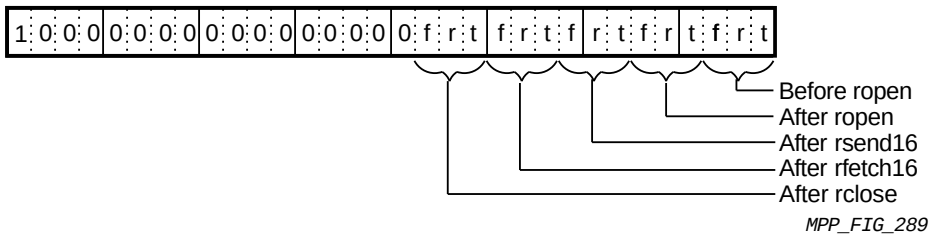
MPP_FIG_287

  - Error message #2:



MPP_FIG_288

  - Error message #3: This record contains the flag bits which were saved during the operation. This indicates that one or more PEs had erroneous flag settings.



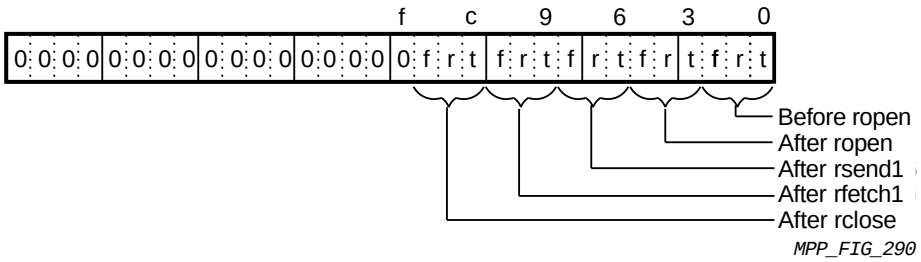
  - Error message #4: This record is an exclusive OR of the correct flag bits with the flag bits as received in message #3. It makes it easier to see which flags were in error. Any bit set here indicates a flag which was in an unexpected state.

- `rt2a`: All PEs Send/Fetch To Self — In this test each PE uses the router to send to and fetch from itself. With all PEs attempting to do this, it takes 16 tries before all the connections have been made and data transferred.

  The test repeats this for 1-bit, 8-bit, 16-bit, 32-bit and 64-bit transfers.

  If it detects an error, the test prints an error message giving the attempted operation, the expected result and the actual result. This is an explanation of the result:

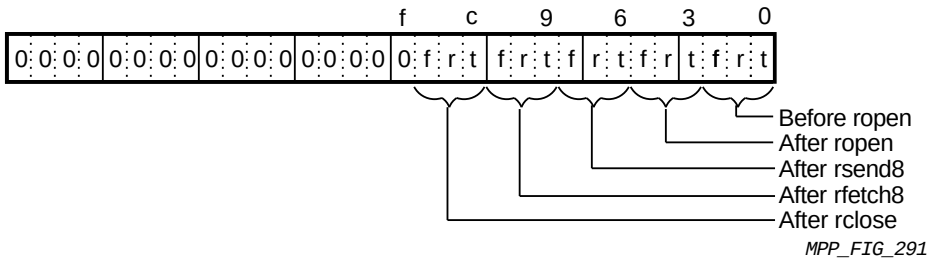
  - Error message #1: This is the number of operations before all PEs had completed their router transfer using the single-bit commands. Only 1/16 of the PEs can be connected at any one time, so it takes 16 operations to complete.
  - Error message #2: This is the exclusive OR comparison of the data sent over the router with the data received at the other end. 16 bits of data were sent using 16 `rsend1` commands. Any bit set in this word indicates an erroneous bit in the data received.
  - Error message #3: This is the exclusive OR comparison of the data fetched over the router with the data received. 16 bits of data were fetched using 16 `rfetch1` commands. Any bit set in this word indicates an erroneous bit in the data received.
  - Error message #4: This is the exclusive OR comparison of the expected flag bits with the actual flag bits set during the rsend and the rfetch operations. The position of the bits in the word indicate which flag is in question and when it was noted. This word does not show the actual state of the flag bits, it only indicates which flags were in error.



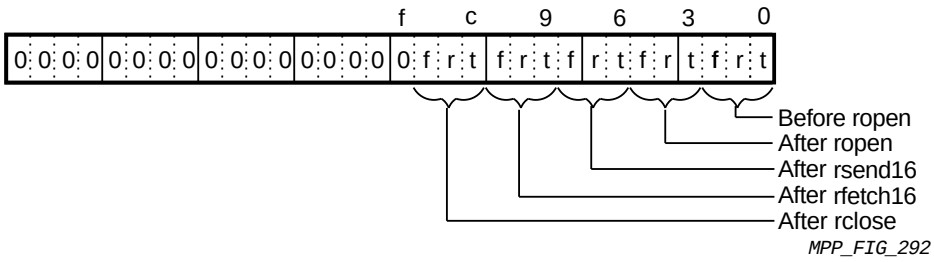
  - Error message #5: This is the number of operations before all PEs had completed their router transfer using the 8-bit commands. Only 1/16 of

the PEs can be connected at any one time, so it takes 16 operations to complete.

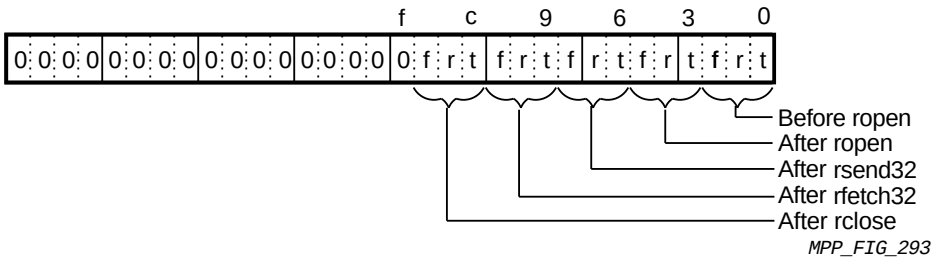
- Error message #6: This is the exclusive OR comparison of the data sent over the router with the data received at the other end. 16 bits of data were sent using two `rsend8` commands. Any bit set in this word indicates an erroneous bit in the data received.
- Error message #7: This is the exclusive OR comparison of the data fetched over the router with the data received. 16 bits of data were fetched using two `rfetch8` commands. Any bit set in this word indicates an erroneous bit in the data received.
- Error message #8: This is the exclusive OR comparison of the expected flag bits with the actual flag bits set during the rsend and the rfetch operations. The position of the bits in the word indicate which flag is in question and when it was noted. This word does not show the actual state of the flag bits, it only indicates which flags were in error.



- Error message #9: This is the number of operations before all PEs had completed their router transfer using the 16-bit commands. Only 1/16 of the PEs can be connected at any one time, so it takes 16 operations to complete.
- Error message #10: This is the exclusive OR comparison of the data sent over the router with the data received at the other end. 16 bits of data were sent using one `rsend16` command. Any bit set in this word indicates an erroneous bit in the data received.
- Error message #11: This is the exclusive OR comparison of the data fetched over the router with the data received. 16 bits of data were fetched using one `rfetch16` command. Any bit set in this word indicates an erroneous bit in the data received.
- Error message #12: This is the exclusive OR comparison of the expected flag bits with the actual flag bits set during the rsend and the rfetch operations. The position of the bits in the word indicate which flag is in question and when it was noted. This word does not show the actual state of the flag bits, it only indicates which flags were in error.

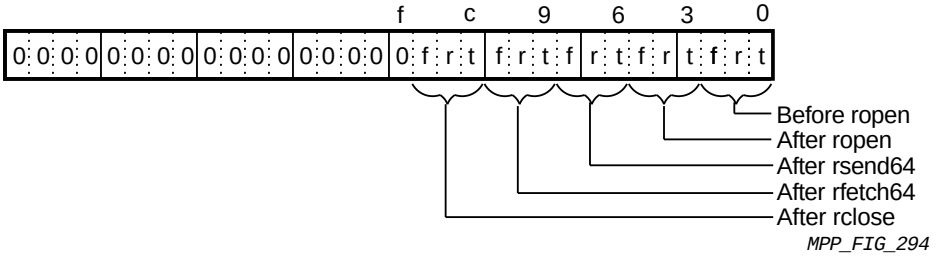


- Error message #13: This is the number of operations before all PEs had completed their router transfer using the 32-bit commands. Only 1/16 of the PEs can be connected at any one time, so it takes 16 operations to complete.
- Error message #14: This is the exclusive OR comparison of the data sent over the router with the data received at the other end. 32 bits of data were sent using one `rsend32` command. Any bit set in this word indicates an erroneous bit in the data received.
- Error message #15: This is the exclusive OR comparison of the data fetched over the router with the data received. 32 bits of data were fetched using one `rfetch32` command. Any bit set in this word indicates an erroneous bit in the data received.
- Error message #16: This is the exclusive OR comparison of the expected flag bits with the actual flag bits set during the rsend and the rfetch operations. The position of the bits in the word indicate which flag is in question and when it was noted. This word does not show the actual state of the flag bits, it only indicates which flags were in error.



- Error message #17: This is the number of operations before all PEs had completed their router transfer using the 64-bit commands. Only 1/16 of the PEs can be connected at any one time, so it takes 16 operations to complete.
- Error message #18: This is the exclusive OR comparison of the data sent over the router with the data received at the other end. 64 bits of data were sent using one `rsend64` command. Any bit set in this word indicates an erroneous bit in the data received. This word represents bits 0-31 of the 64-bit word.
- Error message #19: This word represents bits 32–63 of the 64-bit word discussed in message #18.

- Error message #20: This is the exclusive OR comparison of the data fetched over the router with the data received. 64 bits of data were fetched using one `rfetch64` command. Any bit set in this word indicates an erroneous bit in the data received. This word represents bits 0-31 of the 64-bit word.
- Error message #21: This word represents bits 32–63 of the 64-bit word discussed in message #20.
- Error message #22: This is the exclusive OR comparison of the expected flag bits with the actual flag bits set during the rsend and the rfetch operations. The position of the bits in the word indicate which flag is in question and when it was noted. This word does not show the actual state of the flag bits, it only indicates which flags were in error.



- `rt2b`: All PEs ROSEND And RFETCHC To Self — This test uses the combination command ROSEND to both open the router and send data, and the combination command FETCHC to both fetch data via the router and close the router connection.

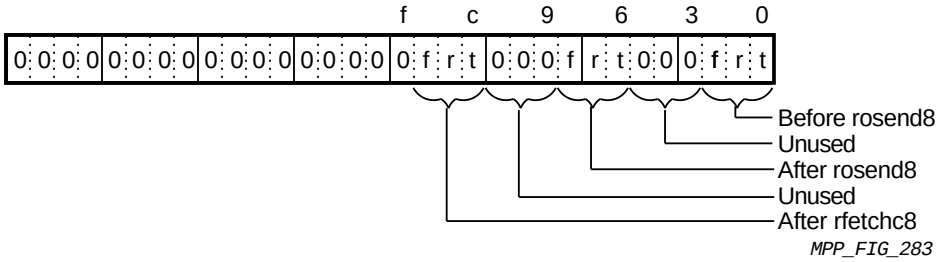
  In this test each PE uses the router to send to and fetch from itself. With all PEs attempting to do this, it takes 16 tries before all the connections have been made and data transferred.

  The test repeats this for 8-bit, 16-bit, 32-bit and 64-bit transfers.

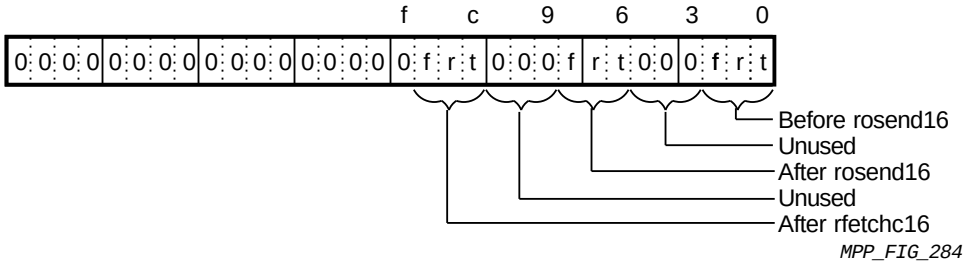
  If it detects an error, the test prints an error message giving the attempted operation, the expected result and the actual result. This is an explanation of the result:

  * Error message #1: This is the number of operations before all PEs had completed their router transfer using the 8-bit commands. Only 1/16 of the PEs can be connected at any one time, so it takes 16 operations to complete.
  * Error message #2: This is the exclusive OR comparison of the data sent over the router with the data received at the other end. 16 bits of data were sent using two `rosend8` commands. Any bit set in this word indicates an erroneous bit in the data received.
  * Error message #3: This is the exclusive OR comparison of the data fetched over the router with the data received. 16 bits of data were fetched using two `rfetchc8` commands. Any bit set in this word indicates an erroneous bit in the data received.
  * Error message #4: This is the exclusive OR comparison of the expected flag bits with the actual flag bits set during the rosend and the rfetchc operations. The position of the bits in the word indicate which flag is in question and when it was noted. This word does not

show the actual state of the flag bits, it only indicates which flags were in error.



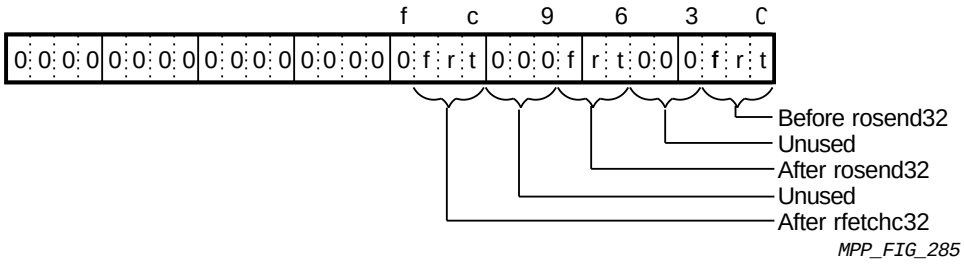
* Error message #5: This is the number of operations before all PEs had completed their router transfer using the 16-bit commands. Only 1/16 of the PEs can be connected at any one time, so it takes 16 operations to complete.
* Error message #6: This is the exclusive OR comparison of the data sent over the router with the data received at the other end. 16 bits of data were sent using one `rosend16` command. Any bit set in this word indicates an erroneous bit in the data received.
* Error message #7: This is the exclusive OR comparison of the data fetched over the router with the data received. 16 bits of data were fetched using one `rfetch16` command. Any bit set in this word indicates an erroneous bit in the data received.
* Error message #8: This is the exclusive OR comparison of the expected flag bits with the actual flag bits set during the rsend and the rfetch operations. The position of the bits in the word indicate which flag is in question and when it was noted. This word does not show the actual state of the flag bits, it only indicates which flags were in error.



* Error message #9: This is the number of operations before all PEs had completed their router transfer using the 32-bit commands. Only 1/16 of the PEs can be connected at any one time, so it takes 16 operations to complete.
* Error message #10: This is the exclusive OR comparison of the data sent over the router with the data received at the other end. 32 bits of data were sent using one `rosend32` command. Any bit set in this word indicates an erroneous bit in the data received.

* Error message #11: This is the exclusive OR comparison of the data fetched over the router with the data received. 32 bits of data were fetched using one `rfetchc32` command. Any bit set in this word indicates an erroneous bit in the data received.

* Error message #12: This is the exclusive OR comparison of the expected flag bits with the actual flag bits set during the rosend and the rfetchc operations. The position of the bits in the word indicate which flag is in question and when it was noted. This word does not show the actual state of the flag bits, it only indicates which flags were in error.



* Error message #13: This is the number of operations before all PEs had completed their router transfer using the 64-bit commands. Only 1/16 of the PEs can be connected at any one time, so it takes 16 operations to complete.

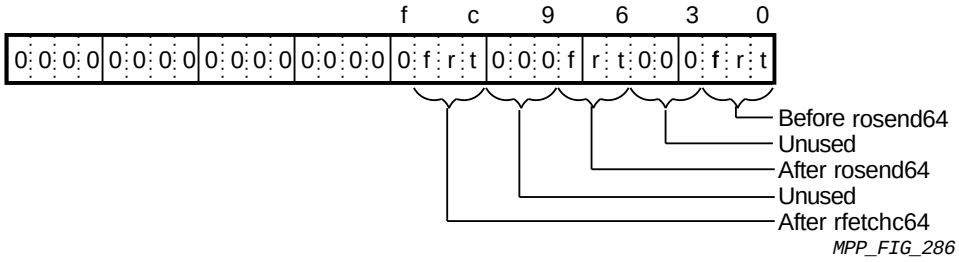
* Error message #14: This is the exclusive OR comparison of the data sent over the router with the data received at the other end. 64 bits of data were sent using one `rosend64` command. Any bit set in this word indicates an erroneous bit in the data received. This word represents bits 0–31 of the 64-bit word.

* Error message #15: This word represents bits 32–63 of the 64-bit word discussed in message #14.

* Error message #16: This is the exclusive OR comparison of the data fetched over the router with the data received. 64 bits of data were fetched using one `rfetchc64` command. Any bit set in this word indicates an erroneous bit in the data received. This word represents bits 0–31 of the 64-bit word.

* Error message #17: This word represents bits 32–63 of the 64-bit word discussed in message #16.

* Error message #18: This is the exclusive OR comparison of the expected flag bits with the actual flag bits set during the rosend and the rfetchc operations. The position of the bits in the word indicate which flag is in question and when it was noted. This word does not show the actual state of the flag bits, it only indicates which flags were in error.



- `rt2c`: Forward Router Send Test — This test asks each PE to make a router send transfer in the forward direction (to a PE whose address is higher). This requires 16 operations before all the contentions have been resolved.

  The test starts with each PE making a send to itself (offset = 0). Then, each PE successively sends data to the PE whose address is offset from its own by a factor of +1 to +511.

  If the test detects an error, it prints an error message giving the router offset at which the error occurred, the nature of the error, the expected and actual result.

- `rt2d`: Forward Router Fetch Test — This test asks each PE to make a router fetch transfer from the forward direction (from a PE whose address is higher). This requires 16 operations before all the contentions have been resolved.

  The test starts with each PE making a fetch from itself (offset = 0). Then, each PE successively fetches data from the PE whose address is offset from its own by a factor of +1 to +511.

  If the test detects an error, it prints an error message giving the router offset at which the error occurred, the nature of the error, the expected and actual result.

- `rt2e`: Reverse Router Send Test — This test asks each PE to make a router send transfer in the reverse direction (to a PE whose address is lower). This requires 16 operations before all the contentions have been resolved.

  The test starts with each PE making a send to itself (offset = 0). Then, each PE successively sends data to the PE whose address is offset from its own by a factor of –1 to –511.

  If the test detects an error, it prints an error message giving the router offset at which the error occurred, the nature of the error, the expected and actual result.

- `rt2f`: Reverse Router Fetch Test — This test asks each PE to make a router fetch transfer from the reverse direction (from a PE whose address is lower). This requires 16 operations before all the contentions have been resolved.

  The test starts with each PE making a fetch from itself (offset = 0). Then, each PE successively fetches data from the PE whose address is offset from its own by a factor of –1 to –511.

If the test detects an error, it prints an error message giving the router offset at which the error occurred, the nature of the error, the expected and actual result.

– `rt2g`: All PEs Send/Fetch To Self (m-machine busy) — In this test each PE uses the router to send to and fetch from itself at the same time that the m-machine is moving data from PReg to PMem. With all PEs attempting to do this, it takes 16 tries before all the connections have been made and data transferred.

  The test repeats this for 1-bit, 8-bit, 16-bit, 32-bit and 64-bit transfers.

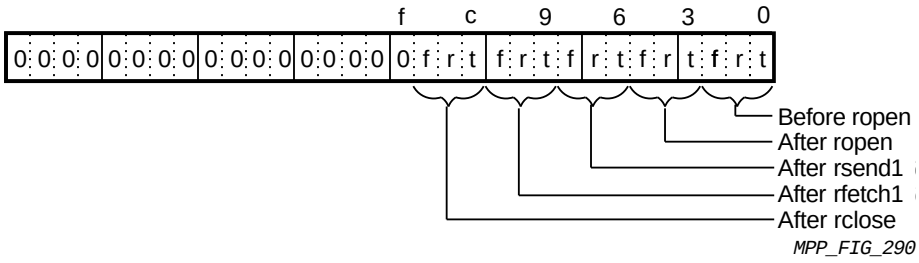
  If it detects an error, the test prints an error message giving the attempted operation, the expected result and the actual result. This is an explanation of the result:

  * Error message #1: This is the number of operations before all PEs had completed their router transfer using the single-bit commands. Only 1/16 of the PEs can be connected at any one time, so it takes 16 operations to complete.

  * Error message #2: This is the exclusive OR comparison of the data sent over the router with the data received at the other end. 16 bits of data were sent using 16 `rsend1` commands. Any bit set in this word indicates an erroneous bit in the data received.

  * Error message #3: This is the exclusive OR comparison of the data fetched over the router with the data received. 16 bits of data were fetched using 16 `rfetch1` commands. Any bit set in this word indicates an erroneous bit in the data received.

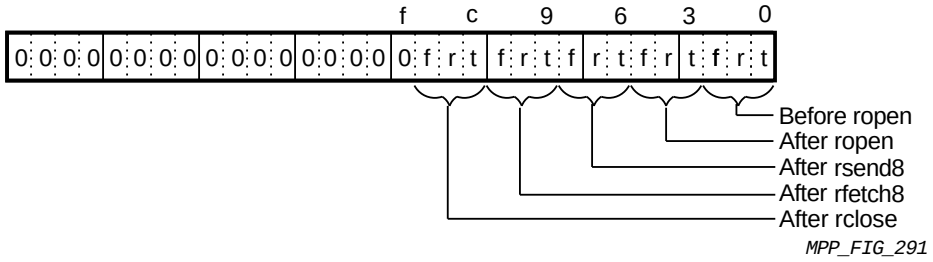
  * Error message #4: This is the exclusive OR comparison of the expected flag bits with the actual flag bits set during the rsend and the rfetch operations. The position of the bits in the word indicate which flag is in question and when it was noted. This word does not show the actual state of the flag bits, it only indicateswhich flags were in error.



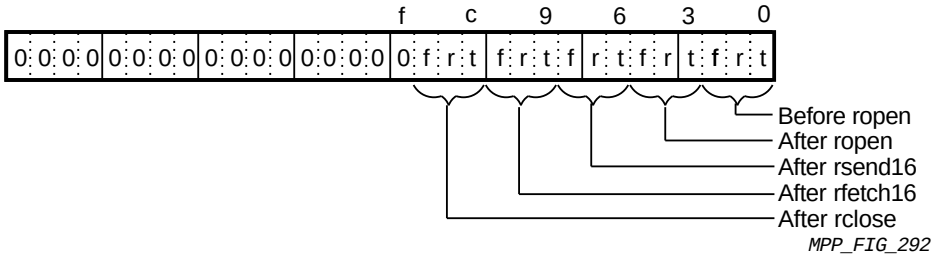
  * Error message #5: This is the number of operations before all PEs had completed their router transfer using the 8-bit commands. Only 1/16 of the PEs can be connected at any one time, so it takes 16 operations to complete.

  * Error message #6: This is the exclusive OR comparison of the data sent over the router with the data received at the other end. 16 bits of data were sent using two `rsend8` commands. Any bit set in this word indicates an erroneous bit in the data received.

* Error message #7: This is the exclusive OR comparison of the data fetched over the router with the data received. 16 bits of data were fetched using two `rfetch8` commands. Any bit set in this word indicates an erroneous bit in the data received.
* Error message #8: This is the exclusive OR comparison of the expected flag bits with the actual flag bits set during the rsend and the rfetch operations. The position of the bits in the word indicate which flag is in question and when it was noted. This word does not show the actual state of the flag bits, it only indicates which flags were in error.



* Error message #9: This is the number of operations before all PEs had completed their router transfer using the 16-bit commands. Only 1/16 of the PEs can be connected at any one time, so it takes 16 operations to complete.
* Error message #10: This is the exclusive OR comparison of the data sent over the router with the data received at the other end. 16 bits of data were sent using one `rsend16` command. Any bit set in this word indicates an erroneous bit in the data received.
* Error message #11: This is the exclusive OR comparison of the data fetched over the router with the data received. 16 bits of data were fetched using one `rfetch16` command. Any bit set in this word indicates an erroneous bit in the data received.
* Error message #12: This is the exclusive OR comparison of the expected flag bits with the actual flag bits set during the rsend and the rfetch operations. The position of the bits in the word indicate which flag is in question and when it was noted. This word does not show the actual state of the flag bits, it only indicates which flags were in error.



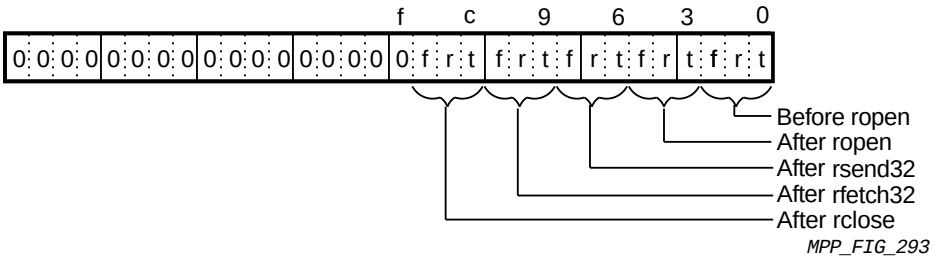
* Error message #13: This is the number of operations before all PEs had completed their router transfer using the 32-bit commands. Only

1/16 of the PEs can be connected at any one time, so it takes 16 operations to complete.

* Error message #14: This is the exclusive OR comparison of the data sent over the router with the data received at the other end. 32 bits of data were sent using one `rsend32` command. Any bit set in this word indicates an erroneous bit in the data received.

* Error message #15: This is the exclusive OR comparison of the data fetched over the router with the data received. 32 bits of data were fetched using one `rfetch32` command. Any bit set in this word indicates an erroneous bit in the data received.

* Error message #16: This is the exclusive OR comparison of the expected flag bits with the actual flag bits set during the rsend and the rfetch operations. The position of the bits in the word indicate which flag is in question and when it was noted. This word does not show the actual state of the flag bits, it only indicates which flags were in error.



* Error message #17: This is the number of operations before all PEs had completed their router transfer using the 64-bit commands. Only 1/16 of the PEs can be connected at any one time, so it takes 16 operations to complete.

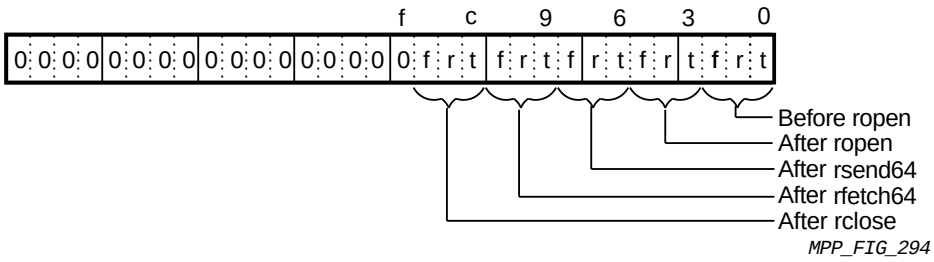
* Error message #18: This is the exclusive OR comparison of the data sent over the router with the data received at the other end. 64 bits of data were sent using one `rsend64` command. Any bit set in this word indicates an erroneous bit in the data received. This word represents bits 0–31 of the 64-bit word.

* Error message #19: This word represents bits 32–63 of the 64-bit word discussed in message #18.

* Error message #20: This is the exclusive OR comparison of the data fetched over the router with the data received. 64 bits of data were fetched using one `rfetch64` command. Any bit set in this word indicates an erroneous bit in the data received. This word represents bits 0–31 of the 64-bit word.

* Error message #21: This word represents bits 32–63 of the 64-bit word discussed in message #20.

* Error message #22: This is the exclusive OR comparison of the expected flag bits with the actual flag bits set during the rsend and the rfetch operations. The position of the bits in the word indicate which flag is in question and when it was noted. This word does not show the actual state of the flag bits, it only indicates which flags were in error.



- `sadd1`: Short ADD Test — This test enables all PEs, then moves two 32-bit numbers and their inverse into PRegs. It first adds the two numbers and then adds the inverse of the two numbers.

  If the test detects an error, it prints an error message giving the attempted operation, the expected result and the actual result. The results have the following meaning:

  * Error message #1: First data word; if this is wrong, one or more PEs could have a problem storing data in PReg 00[c0]. It could also indicate a problem with the GOR32 operation.
  * Error message #2: Second data word; if this is wrong, one or more PEs could have a problem storing data in PReg 32[c0]. It could also indicate a problem with the GOR32 operation.
  * Error message #3: Inverse of first data word; if this is wrong, one or more PEs could have a problem with the XOR32 operation, or trouble storing data in PReg 64[c0].
  * Error message #4: Inverse of second data word; if this is wrong, one or more PEs could have a problem with the XOR32 operation, or trouble storing data in PReg 96[c0].
  * Error message #5: Sum of first and second data word; if messages #1–2 were correct, then one or more PEs are not adding correctly.
  * Error message #6: Sum of first inverse data word and second inverse data word; if messages #3–4 were correct, then one or more PEs are not adding correctly.

- `tagtest0`: Tag Stall Test (Many PRegs To Single PMem Location) — Test of the PReg tag stall. The test performs the following sequence of operations:

  1. It initializes PReg location A with 0x0, and PReg location B with 0x5a.
  2. It stores PReg loc B (0x5a) into a PMem location.
  3. It immediately stores PReg loc A (0x0) into the same PMem location.
  4. It immediately reloads PReg loc A with 0xff. If 0xff is loaded into some of the PMem locations, an error is indicated.
  5. It then attempts to load the PMem location into PReg B. If it gets anything other than zero, it is an error.

  The test repeats for many different PReg locations working into the same PMem location.

The values returned in the error messages may be interpreted as follows:

* Error message #1–#159: Each word is a global OR of the data transferred to and from PMem by each PE. The sequence is:

```
step 1:  0x00 --> 0[c1]
step 2:  0x5a --> 8[c1]
step 3:           8[c1] --- st8 ---> PMem  0x5a
step 4:           0[c1] --- st8 ---> PMem  0x00
step 5:  0xff --> 0[c1]
step 6:  0xa5 --> 8[c1]
step 7:     ? <-- 8[c1] <--- ld8 --- PMem
```

  Each PE should get 0x00 back from PMem, but if the tag stall mechanism is not working, some PEs may get 0x5a back instead:

  - 0x5a: Tag stall mechanism is not working.
  - 0xa5: The LD8 command in step #7 did not return data to destination.
  - 0xff: While the store command in step #4 is still working, step #5 changes the data in 0[c1]. The tag stall mechanism is supposed to prevent this change from affecting the store command still in progress.

* Error message #160: Same as the previous words, except a different addressing mode is used for the PReg address.
* Error message #161: Sanity check to verify that the test can transmit a number other than zero. If this word is zero, then the veracity of all the previous words is suspect.

– `tagtest0f`: Tag Stall Test (Many PRegs To Many PMem Locations) — Fast test of the PReg tag stall. The test performs the following sequence of operations:

1. It initializes PReg location A with 0x0, and PReg location B with 0x5a.
2. It stores PReg loc B (0x5a) into a PMem location.
3. It immediately stores PReg loc A (0x0) into the same PMem location.
4. It immediately re-loads PReg loc A with 0xff. If 0xff is loaded into some of the PMem locations, an error is indicated.
5. It then attempts to load the PMem location into PReg B. If it gets anything other than zero, it is an error.

The test repeats for many different offsets in both PReg and PMem.

If there is an error, the test ends immediately and prints an error message giving the state of the zero flag, the base PMem address where the error occurred, and the base PReg address where the error occurred. It gives no clue as to which PE chip is faulty.

* Error word 1: Expected 0x00000000. This word represents the inverse of the zero flag. Thus, this word is zero if the flag was set (the normal case). This word will be 0x00000001 if the flag was not set (indicating that one or more PEs read a nonzero number from its PMem).

    * Error word 2: Expected 0xffffffff. This number represents the last PMem byte offset the test accessed before ending. During the course of the test this number ranges between 0x3fff and 0x0.
    * Error word 3: Expected 0x4f8. This number represents the last PReg base address the test accessed before ending. During the course of the test, this number ranges between 0x0 and 0x4f8.

- `xnet1`: Shift Data From Odd To Even Columns And Back — This test enables all odd columns to transmit. Each odd PE then transmits its own row/col address as data to its even neighbor to the east a distance of 1. If this operation is successful, all odd PEs have their t-bit set, and all even PEs have their r-bit (received) set.

  It then disables the odd rows and enables the even rows to transmit. Each even row transmits the data received in the previous shift back to its neighbor to the West. Each PE in the odd columns can then compare the data sent with the data received. They should be the same.

  If an error occurs, the test prints an error message containing expected and actual data. This data may be interpreted as follows:

  – Error message #1: 0x00000001 is a flag to indicate that one or more PEs either failed to transmit or receive; or that one or more PEs both transmitted and received. Normally all the PEs in odd columns would transmit, and all the PEs in even columns would receive.
  – Error message #2: 0x00000001 is a flag to indicate that one or more PEs in odd columns did not receive back the same data they sent. Each PE in an odd column sent its own address to its neighbor to the East (the even columns). The neighbor to the East then sent the data back to the original sender. The data should not have changed.
  – Error message #3: Sanity check to verify that the test can indeed handle a number other than zero.

- `xnetE`: XNet Shift East Test — This test selects a single PE at a time and shifts data east a distance which is a function of the PEs address. For each of the 1024 PEs (in a single board system), the test checks the loop count.

  This test is attempting to provoke a parity error. When one occurs, the front-end program times out waiting for more data from the test and prints the status of the HSR register which indicates why the test halted.

  You can still glean some useful information because the front-end program indicates which message record it was waiting for when it timed out. This record number is the distance the test was attempting to shift data when the error occurred. This record number is also the row/column address of the PE which was attempting the shift.

- `xnetN`: XNet Shift North Test — This test selects a single PE at a time and shifts data north a distance which is a function of the PEs address. For each of the 1024 PEs (in a single board system), the test checks the loop count.

  This test is attempting to provoke a parity error. When one occurs, the front-end program times out waiting for more data from the test and prints the status of the HSR register which indicates why the test halted.

You can still glean some useful information because the front-end program indicates which message record it was waiting for when it timed out. This record number is the distance the test was attempting to shift data when the error occurred. This record number is also the row/column address of the PE which was attempting the shift.

- xnetNE: XNet Shift Northeast Test — This test selects a single PE at a time and shifts data northeast a distance which is a function of the PEs address. For each of the 1024 PEs (in a single board system), the test checks the loop count.

  This test is attempting to provoke a parity error. When one occurs, the front-end program times out waiting for more data from the test and prints the status of the HSR register which indicates why the test halted.

  You can still glean some useful information because the front-end program indicates which message record it was waiting for when it timed out. This record number is the distance the test was attempting to shift data when the error occurred. This record number is also the row/column address of the PE which was attempting the shift.

- xnetNW: XNet Shift Northwest Test — This test selects a single PE at a time and shifts data northwest a distance which is a function of the PEs address. For each of the 1024 PEs (in a single board system), the test checks the loop count.

  This test is attempting to provoke a parity error. When one occurs, the front-end program times out waiting for more data from the test and prints the status of the HSR register which indicates why the test halted.

  You can still glean some useful information because the front-end program indicates which message record it was waiting for when it timed out. This record number is the distance the test was attempting to shift data when the error occurred. This record number is also the row/column address of the PE which was attempting the shift.

- xnetS: XNet Shift South Test — This test selects a single PE at a time and shifts data south a distance which is a function of the PEs address. For each of the 1024 PEs (in a single board system), the test checks the loop count.

  This test is attempting to provoke a parity error. When one occurs, the front-end program times out waiting for more data from the test and prints the status of the HSR register which indicates why the test halted.

  You can still glean some useful information because the front-end program indicates which message record it was waiting for when it timed out. This record number is the distance the test was attempting to shift data when the error occurred. This record number is also the row/column address of the PE which was attempting the shift.

- xnetSE: XNet Shift Southeast Test — This test selects a single PE at a time and shifts data SouthEast a distance which is a function of the PEs address. For each of the 1024 PEs (in a single board system), the test checks the loop count.

  This test is attempting to provoke a parity error. When one occurs, the front-end program times out waiting for more data from the test and prints the status of the HSR register which indicates why the test halted.

You can still glean some useful information because the front-end program indicates which message record it was waiting for when it timed out. This record number is the distance the test was attempting to shift data when the error occurred. This record number is also the row/column address of the PE which was attempting the shift.

- xnetSW: XNet Shift Southwest Test — This test selects a single PE at a time and shifts data southwest a distance which is a function of the PEs address. For each of the 1024 PEs (in a single board system), the test checks the loop count.

  This test is attempting to provoke a parity error. When one occurs, the front-end program times out waiting for more data from the test and prints the status of the HSR register which indicates why the test halted.

  You can still glean some useful information because the front-end program indicates which message record it was waiting for when it timed out. This record number is the distance the test was attempting to shift data when the error occurred. This record number is also the row/column address of the PE which was attempting the shift.

- xnetW: XNet Shift West Test — This test selects a single PE at a time and shifts data west a distance which is a function of the PEs address. For each of the 1024 PEs (in a single board system), the test checks the loop count.

  This test is attempting to provoke a parity error. When one occurs, the front-end program times out waiting for more data from the test and prints the status of the HSR register which indicates why the test halted.

  You can still glean some useful information because the front-end program indicates which message record it was waiting for when it timed out. This record number is the distance the test was attempting to shift data when the error occurred. This record number is also the row/column address of the PE which was attempting the shift.

Options

-b
Use this option to specify Burn-in test; runs the diagnostic repetitively, reporting the error count at the end of each pass.

-q
Use this option to specify Quick test; selects a brief version of some of the tests.

-t
Use this option to specify Terse message style; the test prints only the most essential messages.

Files

Executable file found in directory $MP_PATH/field/bin: pe_macro

See Also

pe_diag, pe_rtbp, pe_scan, pe_xnet

pe_memdiag(1)

pe_memdiag — DECmpp Sx indirect load/store tests

Syntax

pe_memdiag [-bqt]

Description

The pe_memdiag command loads and runs a back-end program which does the following operations:

1. Has each processor element (PE) do 1000 (hex) indirect stores followed by 1000 indirect loads; checks for error
2. Has each PE do 1000 direct loads of the same data; checks for error
3. Has each PE do 1000 indirect stores followed by 1000 inverted indirect loads; checks for error
4. Has each PE do 1000 inverted direct loads of the the same data; checks for error

If there are no errors, the program returns a success code and halts. If there are errors, the program returns the following:

- Operation type (1–4 above)
- Direct address involved with the error
- GOR of indirect address involved with the error
- Operation type (Report on individual PE)
- PE number
- Indirect address
- Data received
- Data expected

The program checks all PEs and returns a separate report for each one found to be in error.

Options

-b
Use this option to specify the burn-in test; runs the diagnostic repetitively, reporting the error count at the end of each pass.

-t
Use this option to specify terse message style; prints only the most essential messages.

Files

$DIAG_PATH/field/bin/pe_memdiag

pe_rtbp(1)

pe_rtbp — DECmpp Sx router backplane test

Syntax

pe_rtbp [-qtb] [1100]

Description

The pe_rtbp command tests various signal paths which pass between processor element (PE), backplane and router cards. When the test detects a failing path, it gives a complete report starting with the source board, chip and pin number, the signal name as shown on the schematic, the backplane connector and pin number, the destination board, backplane connector and pin number, and the destination chip and pin number.

The verbose message format is self-explanatory. The terse format for experienced users is shown in the following example:

```
BD04 U02RS1-127 MQ20 P0114E-12 -- Bp_S1_0400_2 -- BD00 U02RS2-28 MD14 P0114E-7
```

- BD04: Source board slot number.
- U02RS1-127: Source chip and pin number.
- MQ20: Source pin name.
- P0114E-12: Source backplane connector and pin number.
- Bp_S1_0400_2: Backplane signal name.
- BD00: Destination board slot number.
- U02RS2-28: Destination chip and pin number.
- MD14: Destination pin name.
- P0114E-7: Destination backplane connector and pin number.

Options

1100
Use this option to adjust the test for a DECmpp 12000–LC model (4 card slots). The default machine is the DECmpp 12000 (16 card slots).

-b
Use this option to specify the burn-in test; runs the diagnostic repetitively, reporting the error count at the end of each pass.

-q
Use this option to specify quick test. This selects a brief version of some of the tests.

-t
Use this option to specify terse message style. The test prints only the most essential messages.

Files

Executable binary: `$MP_PATH/field/bin/pe_rtbp`

pe_rtdiag(1)

pe_rtdiag — DECmpp Sx processor element (PE) router diagnostic

Syntax

pe_rtdiag [-t]

Description

The pe_rtdiag command uses a back-end program controlled by an HDB script. It performs three test routines:

- Identity pattern: Using the router, each PE shifts data to itself
- Multiplex pattern: Using the router, PE #0 shifts data to all PEs one at a time
- Demultiplex pattern: Using the router, all PEs shift data to PE #0 one at a time

After each shift, the back-end program checks that the source T-bit is no longer set, that the destination R-bit is set and that the correct data was received at the destination.

If an error occurs, the HDB script prints the board, chip and cluster information of the offending PE. It also prints the state of the applicable error bits in the HSR and PPSW registers.

Options

-t
Use this option to specify terse message style; prints only the most essential messages.

Files

Binary executable file: $DIAG_PATH/field/bin/pe_rtdiag

pe_rtr(1)

pe_rtr — DECmpp Sx processor element (PE) router diagnostic

Syntax

pe_rtr [-t]

Description

The pe_rtr command uses a back-end program controlled by an HDB script. It shifts data and checks whether it is received. It has each PE keep its own error counter, which it polls at the end of the test (by peek/poking from HDB).

Options

-t
Use this option to specify terse message style. The test prints only the most essential messages.

Files

Binary executable file: $DIAG_PATH/field/bin/pe_rtr

pe_scan(1)

pe_scan — DECmpp Sx serial scan test

Syntax

pe_scan [-qtb]

Description

The pe_scan command tests the serial scan chains on the array control unit (ACU) board, the processor element (PE) board and the router boards:

- ACU board
  - Main scan chain
  - EEPROM scan chain
- PE and Router boards
  - Router shift chain S1, S2 and S3
  - EEPROM scan chain
  - GOR scan chain
  - PREG scan chain
  - Parity:
    * Preg address parity
    * PE instruction parity
    * RT instruction parity
    * M-machine instruction parity

Options

-b
Use this option to specify burn-in test; runs the diagnostic repetitively, reporting the error count at the end of each pass.

-q
Use this option to specify the quick test. This selects a brief version of some of the tests.

-t
Use this option to specify terse message style. The test prints only the most essential messages.

Files

$MP_PATH/field/bin/pe_scan

dpumanager(8)

dpumanager — DECmpp Sx data parallel unit (DPU) job manager daemon, Version 1.1

Syntax

etc/dpumanager [*options...*]

Description

The DECmpp Sx job manager daemon, dpumanager, maintains the queue for data parallel unit (DPU) jobs, determines which job has access to the DPU at any particular time, and ensures that the DPU state is properly initialized between jobs. It uses a privileged access mode to the array control unit (ACU) driver in order to monitor calls to open(2) and close(2) and certain calls to ioctl(2). This allows it to determine which processes need access to the DPU.

When started, dpumanager puts itself in the background by forking a child process and exiting the parent. This behavior may be suppressed by using the -nodaemon option.

Jobs are queued in order of priority and the time when the DPU was requested. The job manager assigns a queue priority based primarily on the job's processor element (PE) memory requirement (refer to mplimit(1)). In addition, jobs with a time limit of one minute or less get a small boost in priority. The priority is an integer from 0 to 100, where a lower number is a higher priority (refer to mpq(1)). A job is queued in front of the first job that has a lower priority. A job's priority increases every time another (higher priority) job skips in front of it, so it is not possible for small jobs to permanently block execution of a large job.

The first *n* jobs on the queue are loaded in DPU memory and share the machine in a round-robin fashion. The number *n* varies according to job requirements and memory availability. The maximum value for *n* is determined by the -jobs command line option.

The round-robin scheduling involves cooperation between the job manager and the mppehook or mppeback process. The mppehook/mppeback process informs the job manager of program requirements. The job manager informs mppehook /mppeback when a job's time slice is over. The mppehook/mppeback process informs the job manager when the job is in a quiescent state and can be safely swapped out. Jobs that do not run under mppehook or mppeback are not able to share the DPU. Note that all user jobs normally run under mppehook or mppeback. They run under mppeback if the user explicitly invokes the DECmpp Sx Parallel Programming Environment (MPPE) using the mppe command and then invokes the program from within that user interface; otherwise, they run under mppehook until they fault. If you use the ps command, you see the user's executable listed twice: the mppehook or mppeback process is the lower numbered of the two user processes of the job, and this process number is the same as the number reported by mpq. The child process, which is executing the user's code, has a higher process number.

The number of jobs in memory is greater than one only when multiple jobs at the front of the queue are able to share the machine. A job cannot share the machine if it needs all of memory or it is not running under mppehook or mppeback (user jobs normally run under one of these two processes). If a job cannot share the

machine, it must wait until it reaches the head of the queue before it is loaded into memory, and then it has exclusive access until it terminates.

To get on the queue, it is necessary to make an ioctl(2) call, DPUIOACCESS, to the ACU driver. This is normally done transparently by program startup routines in the DECmpp Sx-provided libraries. This ioctl(2) call does not return until the program is granted access.

When access to the DPU is granted, the job manager assigns one or more partitions of PE memory. The amount assigned is normally determined by fields in the object file header, which are set using the mplimit(1) command. The memory allocation cannot be dynamically increased, so it is important that the job request an adequate quantity. The default is determined by the -pmem option.

The job manager attempts to assure that an appropriate version of DPU microcode (and opcode map) is loaded for each job. When started, it loads the default microcode version from $MP_PATH/etc/mp*ucode.wo. At the beginning of each job it reloads the microcode if the wrong version is known to be loaded or if the machine does not respond as expected.

The job manager is also responsible for loading and starting up the ACU kernel program. The ACU kernel is reloaded whenever microcode is reloaded or whenever the kernel does not appear to be operating correctly.

Jobs may have a time limit set. A system maximum time limit is set by the -maxtime command line option, by the privileged ioctl(2) call, DPUIOSYSTIMELIMIT, or using the command mptimelimit(8). The default is no time limit. When a time limit is exceeded, the job manager sends the offending process a SIGXCPU signal, which normally causes mppehook/mppeback and the user process to exit.

The job manager accounting file (/usr/adm/dpuacct) contains an entry for the beginning and end of every job. This binary file is read by mpstat(1).

The job manager maintains a shared memory segment containing the job queue structures. This may be examined using the mpq(1) command.

The job manager also maintains a DPU configuration file (/usr/tmp/.dpuconfig). This is used by mpi(1) and mpd(8) to determine the configuration of DPU systems on the local network.

The terminate (15) signal causes the job manager to send a hangup signal to all the jobs and then exit. The hangup (1) signal tells it to send a hangup signal to all the jobs but not exit. The quit (3) signal tells it to send a hangup signal just to the current active job.

Environment

The job manager uses the environment variable $MP_PATH to find the default microcode image ($MP_PATH/etc/mp*ucode.wo) and the ACU kernel ($MP_PATH/etc/acuk).

Options

-jobs *n*
Use this option to specify the maximum number of jobs in memory and the number of memory partitions. The valid range is 1 to 16. The default is 1, which disables job swapping. A value of 4 is recommended when job swapping is desired.

-maxtime *t*
Use this option to specify a system maximum time limit (*t*), in seconds, for all jobs.

-nodaemon
Use this option to prevent the job manager from putting itself in background.

-pmem *k*
Use this option to specify the default amount of PE memory (in KB) that each job is assigned. The default is one partition, determined by the -jobs option.

-slice *t*
Use this option to specify the amount of time allocated to a time slice, in milliseconds. The default is 10000 (10 seconds). *Warning: a value of less than 3000 (3 seconds) will most likely result in thrashing.*

-Zq
Use this option to suppress printing of the copyright notice.

Restrictions

Pending jobs hang if the job manager terminates abnormally. In this instance, it is necessary to kill all such jobs manually; they will never be granted access to the DPU even when the job manager is restarted.

The job manager must always be executed with root privileges.

Diagnostics

The job manager log (`/usr/adm/dpujobmgr.log`) is a plain-text file containing a time-stamped entry for each significant event detected by the job manager. All faults detected by the job manager or the background diagnostic process are reported in this log file. In addition, an entry is made when the job manager starts up, shuts down, or loads microcode.

When a fault is detected that requires termination of a job, the job manager sends an IOT (6) signal to the mppehook or mppeback process that is controlling that job. If there is no mppehook/mppeback process, a hangup (1) signal is used.

Files

`/usr/adm/dpujobmgr.log` — Log file
`/usr/adm/dpuacct` — Job accounting file
`/usr/tmp/.dpuconfig` — DPU configuration file
`$MP_PATH/etc/acuk` — ACU kernel
`$MP_PATH/etc/mp11ucode.wo` — Default microcode image for DECmpp 12000–LC
`$MP_PATH/etc/mp12ucode.wo` — Default microcode image for DECmpp 12000
`$MP_PATH/etc/startup.wo` — Dummy microcode used during initialization

See Also

`mpi(1)`, `mpq(1)`, `mplimit(1)`, `mpstat(1)`, `dpuDevice(3)`, `acu(4)`, `mpd(8)`, `mptimelimit(8)`

mpshutdown(8)

mpshutdown — Terminates the DECmpp Sx data parallel unit (DPU) job manager, Version 1.1

Syntax

etc/mpshutdown

Description

The mpshutdown command sends a termination signal to the data parallel unit (DPU) job manager daemon, dpumanager(8). When the job manager receives this signal, it should send a hangup signal to all pending DPU jobs and then exit.

This is a shell script which should be run as root.

Diagnostics

The mpshutdown command exits with a nonzero code if:

- It is unable to kill the job manager.
- The job manager is not present.
- It finds multiple instances of the job manager.

See Also

dpumanager(8)

Index

E

F

H

I

J

K

L

M

P

Q

R

S

T

U

V

X