Tru64 UNIX

System Configuration Supplement: OEM Platforms

Part Number: AA-RJ1UC-TE

April 2000

Product Version: Tru64 UNIX Version 5.0A or higher

This manual provides information needed to set up OEM platforms running the HP Tru64 UNIX operating system. It helps system and network administrators configure PCI/ISA modular single-board computers (SBCs), Alpha VME SBCs, and VMEbus backplane (vb) networks in which SBCs operate as Ethernet nodes.

Hewlett-Packard Company
Palo Alto, California

Contents

2 Configuring the VMEbus for Alpha VME Systems

3 Configuring a VMEbus Backplane (vb) Network

Index

Tables

About This Manual

This manual provides information needed to set up OEM platforms running the HP Tru64 UNIX operating system. It helps system and network administrators configure PCI/ISA modular single-board computers (SBCs), Alpha VME SBCs, and VMEbus backplane (`vb`) networks in which SBCs operate as Ethernet nodes.

Audience

This manual is for experienced system and network administrators who are thoroughly familiar with their platform's I/O bus and with the operating system concepts, commands, and configurations.

Organization

This manual contains the following chapters.

Chapter 1 OEM Platform Requirements and Restrictions
Provides notes about the use of OEM platforms, with a section devoted to each platform family.

Chapter 2 Configuring the VMEbus for Alpha VME Systems
Explains how to configure VMEbus adapters for OEM platforms, with a section devoted to each major adapter type.

Chapter 3 Configuring a VMEbus Backplane (vb) Network
Explains how to set up a VMEbus backplane–based network in which Alpha VME single-board computers (SBCs) operate as Ethernet nodes.

Related Documents

The following documents are relevant to setting up OEM platforms:

- *System Configuration and Tuning*
- *System Administration*
- *Network Administration: Connections*
- *Network Administration: Services*
- Your platform's hardware documentation
- The `sys_attrs_vba_vipvic`(7) kernel subsystem reference page

- The `sys_attrs_vba_univ`(7) kernel subsystem reference page
- The `sys_attrs_vme_vba`(7) kernel subsystem reference page
- The `sys_attrs`(5) reference page
- The `sysconfigdb`(8) reference page
- *Release Notes* Processor-Specific Notes
- *Installation Guide* platform-specific instructions for booting
- *Guide to Realtime Programming*
- Device Driver Kit manual *Writing VMEbus Device Drivers*
- Device Driver Kit manual *Writing PCI Bus Device Drivers*

Icons on Tru64 UNIX Printed Manuals

The printed version of the Tru64 UNIX documentation uses letter icons on the spines of the manuals to help specific audiences quickly find the manuals that meet their needs. (You can order the printed documentation from HP.) The following list describes this convention:

G	Manuals for general users
S	Manuals for system and network administrators
P	Manuals for programmers
R	Manuals for reference page users

Some manuals in the documentation help meet the needs of several audiences. For example, the information in some system manuals is also used by programmers. Keep this in mind when searching for information on specific topics.

The *Documentation Overview* provides information on all of the manuals in the Tru64 UNIX documentation set.

Reader's Comments

HP welcomes any comments and suggestions you have on this and other Tru64 UNIX manuals.

You can send your comments in the following ways:

- Fax: 603-884-0120 Attn: UBPG Publications, ZKO3-3/Y32
- Internet electronic mail: `readers_comment@zk3.dec.com`

  A Reader's Comment form is located on your system in the following location:

  `/usr/doc/readers_comment.txt`

Please include the following information along with your comments:

- The full title of the manual and the order number. (The order number appears on the title page of printed and PDF versions of a manual.)
- The section numbers and page numbers of the information on which you are commenting.
- The version of Tru64 UNIX that you are using.
- If known, the type of processor that is running the Tru64 UNIX software.

The Tru64 UNIX Publications group cannot respond to system problems or technical support inquiries. Please address technical questions to your local system vendor or to the appropriate HP technical support office. Information provided with the software media explains how to send problem reports to HP.

Conventions

This manual uses the following conventions:

%	A percent sign represents the C shell system prompt.
#	A number sign represents the default superuser prompt.
>>>	Three right angle brackets represent the console subsystem prompt.
% **cat**	Boldface type in interactive examples indicates typed user input.
file	Italic (slanted) type indicates variable values, placeholders, and routine argument names.
⋮	A vertical ellipsis indicates that a portion of an example that would normally be present is not shown.
cat(1)	A cross-reference to a reference page includes the appropriate section number in parentheses. For example, cat(1) indicates that you can find information on the cat command in Section 1 of the reference pages.

1

OEM Platform Requirements and Restrictions

This chapter provides notes about the use of OEM platforms, with a section devoted to each platform family:

- PCI/ISA modular single-board computers [SMARTengine/Alpha and EBMnn] (Section 1.1)
- Alpha VME 4/nnn and 5/nnn single-board computers [EBVnn] (Section 1.2)
- AXPvme single-board computers (Section 1.3)

1.1 PCI/ISA Modular Single-Board Computers (SMARTengine/Alpha and EBMnn)

The SMARTengine/Alpha 21264 single-board computer (SBC) and its predecessors, the EBM2n and EBM4n SBCs, are members of a family of PCI/ISA-based modular computing components. (The PCI/ISA systems and components product family was formerly known as DIGITAL Modular Computing Components, or DMCC).

The SMARTengine/Alpha 21264 PCI/ISA SBC is a PICMG-compliant processor card based on the Alpha 21264 CPU. The EBM2n and EBM4n SBCs are PICMG-compliant processor cards based on the Alpha 21164 and 21064A CPUs, respectively.

The following notes are specific to PCI/ISA modular SBCs.

1.1.1 Verifying CPU Version

You can use the `sizer` utility to identify SMARTengine/Alpha 21264, EBM2n, and EBM4n SBCs. The `sizer -c` command displays the following output for SMARTengine/Alpha 21264 SBCs:

```
sysname> sizer -c
cpu      "DMCCEV6"
```

The `sizer -c` command displays the following output for EBM2n SBCs:

```
sysname> sizer -c
cpu      "DECEV56_PBP"
```

The `sizer -c` command displays the following output for EBM4n SBCs:

```
sysname> sizer -c
cpu      "DECEV45_PBP"
```

1.1.2 Firmware Requirements

Before installing the operating system, make sure that your system has the correct firmware version. The minimum firmware version required for SMARTengine/Alpha 21264 SBCs is Version 5.6-6903 or higher. The minimum firmware version required for EBM2n and EBM4n SBCs is Version 4.7 or higher. If you have an earlier firmware version, update your firmware before installing the operating system software. For information on how to update your firmware, see the firmware documentation.

To determine the version of firmware on your system, enter the following console firmware command at the prompt:

```
>>> show version
```

1.1.3 Installing Tru64 UNIX

For information about installing the operating system on a SMARTengine/Alpha 21264, EMB2n, or EBM4n SBC, see the Tru64 UNIX *Installation Guide*. The *Installation Guide* provides platform-specific instructions for booting. For the SMARTengine/Alpha 21264 SBC, follow the same instructions as for the EBM2n and EBM4n SBCs.

1.1.4 Restrictions and Known Problems

The following restrictions and known problems apply to PCI/ISA modular SBCs.

1.1.4.1 Option Card Restrictions

You can use the SMARTengine/Alpha 21264, EBM2n, and EBM4n SBCs on PCI/ISA backplanes in the ETMXB/ETMAB family and in corresponding kernels (platforms) in the ETMnn family. Table 1–1 lists the currently supported PCI/ISA backplanes and kernels. Not every SBC is supported in every backplane and kernel; see the current PCI/ISA components order configuration guide for details.

Table 1–1: Supported PCI/ISA Backplanes and Kernels

Backplane	Kernel	Description
ETMXB-BA	ETM05-xx	5-slot PICMG (2 PCI, 1 PCI/ISA, 1 ISA, 1 SBC)
ETMXB-DA	ETM27-SA, 3X-ETM17-xx	7-slot PICMG (3 PCI, 1 PCI/ISA, 1 ISA, 2 SBC [1 SBC slot usable at a time])
ETMAB-CA	ETM25-xx, 3X-ETM15-xx	10-slot PICMG (6 PCI, 1 PCI/ISA, 1 ISA, 2 SBC [1 SBC slot usable at a time])
ETMAB-EA	ETM29-xx, 3X-ETM19-xx	10-slot PICMG (4 PCI/ISA, 4 ISA, 2 SBC [1 SBC slot usable at a time])
ETMAB-AB	ETM31-CA	14-slot PICMG (7 PCI, 6 ISA, 1 SBC)
ETMAB-BB	ETM33-CA	14-slot PICMG (10 PCI, 3 ISA, 1 SBC)
ETMAB-AC	ETM42-CA	19-slot PICMG (10 PCI, 7 ISA, 2 SBC [1 SBC slot usable at a time])
ETMAB-BC	ETM44-CA	19-slot PICMG (13 PCI, 4 ISA, 2 SBC [1 SBC slot usable at a time])

Table Note

All ETMAB backplanes use PCI-to-PCI bridge (PPB) technology to provide both primary (in front of the bridge) and secondary (behind the PPB) slots. All ETMAB backplanes are compliant with PCI Version 2.1.

The option cards shown in Table 1–2, in addition to working in front of the bridge, work behind the bridge. You can plug these cards into any available slot.

Table 1–2: PCI/ISA Options Supported Behind the Bridge

Option Type	Part Number	Description
Graphics	SN-PBXGB-AA	TGA2 2MB PowerStorm 3D30
Graphics	SN-PBXGK-BB	Elsa GLoria Synergy
SCSI	KZPBA-CB	Qlogic PCI Ultra Wide differential SCSI controller
SCSI	KZPCM-DA	Dual-channel PCI to Ultra SCSI adapter with Ethernet controller
SCSI	KZPSA-BB	PCI differential SCSI adapter
SCSI	SN-KZPBA-CA	Qlogic PCI-SCSI Ultra Wide adapter (supports both narrow and wide drives)

Table 1–2: PCI/ISA Options Supported Behind the Bridge (cont.)

Option Type	Part Number	Description
SCSI	KZPAA-AA	PCI-SCSI host bus adapter
Network	DE450-CA	PCI NIC (TP, TW, AUI)
Network	DE500-BA	PCI NIC (TP)

Table Notes

- The SN-PBXGB-AA (TGA2 PowerStorm 3D30) video card will work behind a bridge in multiple configurations if the first card is within the primary bus. For restrictions on jumper settings and X server DMA for the PowerStorm 3D30 card, see Section 1.1.4.2.
- When used with EBM2n SBCs, the SN-KZPBA-CA (PCI-SCSI Ultra Wide adapter) requires the following console parameter to be set:

  ```
  >>> set pci_prefetch SMS
  ```

1.1.4.2 PBXGB-AA (TGA2 PowerStorm 3D30) Video Card Restrictions

The following restrictions apply to the PBXGB-AA (TGA2 PowerStorm 3D30) video card (listed in Table 1–2).

1.1.4.2.1 EV5 Alias Jumper Setting (SMARTengine/Alpha 21264 and EBM2n Only)

For SMARTengine/Alpha 21264 and EBM2n SBCs only, you must set the EV5 Alias jumper on the PowerStorm 3D30 card to On.

1.1.4.2.2 VGAEN Jumper Settings

Only one PowerStorm 3D30 card can have its VGAEN jumper set to On. This card must be positioned in a primary PCI slot for the SRM Console to be displayed. All other PowerStorm 3D30 cards must have their VGAEN jumpers set to Off but may be positioned in any PCI slot. For more information about the jumpers, see the *PBXGB-AA/CA PCI Graphics Option Owner's Guide*, provided with the card.

1.1.4.2.3 X Server DMA Writes Must Be Disabled for Some Configurations

Some configurations of PowerStorm 3D30 cards on SMARTengine/Alpha 21264, EBM2n, and EBM4n SBCs require that you disable X server direct memory access (DMA) write operations. Specifically, you must disable these operations if the system contains multiple PowerStorm 3D30 cards, or if

any PowerStorm 3D30 graphics card is installed behind the PCI-to-PCI bridge (PPB).

The general procedure for disabling X server DMA write operations is as follows:

1. Bring the system to single-user mode.

   If you are able to use the `shutdown` command, execute the following command as superuser:

   ```
   # /usr/sbin/shutdown +2 "Disabling graphics DMA writes"
   ```

   If you cannot use the `shutdown` command (for example, if the X server on the video card is hung), you must halt your system by pressing the hardware halt button and then reboot your system to single-user mode by entering the following command:

   ```
   >>> boot -fl s
   ```

2. Mount all local file systems.

   After your system is in single-user mode, mount all of your local file systems by entering the following command:

   ```
   # bcheckrc
   ```

3. Change the directory to `/usr/var/X11` by entering the following command:

   ```
   # cd /usr/var/X11
   ```

4. Save a copy of the `Xserver.conf` file by entering a command such as the following:

   ```
   # cp Xserver.conf Xserver.conf.old
   ```

5. Edit the `Xserver.conf` file to add the text `-I -ffbDoDMA 4` to the command line arguments section. For example, if the command line arguments section is in its initial default state, it appears as follows:

   ```
   ! you specify command line arguments here
   args <
           -pn
   >
   ```

   Insert the text `-I -ffbDoDMA 4` after `-pn` as follows:

   ```
   ! you specify command line arguments here
   args <
           -pn -I -ffbDoDMA 4
   >
   ```

6. Return the system to multiuser mode by executing the following command:

   ```
   # init 3
   ```

With this change, the video card and X server will run correctly on the SBC in multiuser mode.

1.1.4.3 Operator Control Panel and Watchdog Timer Supported Only in Hardware and Firmware

The operating system does not support the operator control panel or watchdog timer. These server management features are supported only in the hardware and the firmware.

1.1.4.4 IDE Device Mapping Potentially Impacts 21264 SBC Upgrades

The operating system identifies the IDE controllers on the SMARTengine/Alpha 21264 SBC as SCSI devices, which affects the naming of all other SCSI devices in the system. Even though the operating system does not support IDE drives on the 21264 SBC, the IDE controllers are configured during the system boot, causing the disk numbering to be shifted as if two SCSI controllers were added to the configuration.

This is not a significant issue for deploying new systems on the 21264 SBC or for SBC upgrades performed with a new operating system installation, but it can cause problems for SBC upgrades performed without a new operating system installation.

The altered naming of SCSI devices can create problems with `/etc/fstab` file entries and Logical Storage Manager (LSM) features that rely on a previous installation's device naming.

After a 21264 SBC upgrade, if the existing system disk has been renumbered (for example, from `rz0` to `rz16`), the existing system will not boot from the existing system disk. The root, `usr`, and swap partitions to which `fstab` points no longer exist. To resolve the problem, you must edit the `fstab` file, changing device name references (for example, from `rz0` to `rz16`). As the swap partition is not accessible, the root partition cannot be made writable. Thus you must modify the `fstab` file before the existing system is upgraded, or you must boot the Tru64 UNIX distribution CD-ROM in single-user mode to edit the file.

If LSM features were used in connection with the existing operating system installation, further steps may be necessary. After a 21264 SBC upgrade, LSM volume data on any renumbered disk no longer matches the physical configuration. In particular, if a system disk containing LSM volumes is renumbered, changes similar to the following will be required before the upgraded system will boot into multiuser mode:

1. Before the SBC upgrade, disable LSM volumes on the system disk; see the `volunroot -a` command in the `volunroot`(8) reference page. You

must also edit `/etc/fstab` to remove the LSM mount point. (See the `fstab`(4) reference page.)

2. Update `/etc/fstab` entries to reflect device name changes resulting from the SBC upgrade. As previously mentioned, you must make these changes either before the SBC upgrade or while booted in single-user mode from the operating system CD-ROM.
3. After the SBC upgrade, reconvert disk partitions on the system disk to LSM volumes as desired. (See the `volencap`(8) reference page.)

1.1.5 Configuring PCI/ISA Modular 8-Headed Graphics Systems

This section describes how to configure a PCI/ISA modular system to run 8-headed graphics.

You can configure PCI/ISA platforms that contain a EBM2n-AZ Alpha PICMG single-board computer (SBC) and multiple PowerStorm 3D30 graphics cards to run multiheaded graphics, controlling up to eight monitors at a time.

1.1.5.1 Hardware and Software Requirements

Running 8-headed graphics requires the following:

- An EBM2n-AZ Alpha PICMG SBC and eight PowerStorm 3D30 graphics cards within a fully configured PCI/ISA system.
- A PCI/ISA backplane and enclosure with at least 10 PCI slots, 512 MB main memory, a supported Ethernet card, and all the other storage and I/O options normally required for such a system. (See the current PCI/ISA components order configuration guide.)
- Correct card placement: the SBC occupies an SBC slot and the graphics cards occupy eight PCI slots, in the order described in Section 1.1.5.2.
- Version 4.0E or higher of the operating system.
- The latest DMCC SRM code from Version 5.2 or higher of the Firmware CD-ROM.

The following PCI/ISA system configuration has been qualified for running 8-headed graphics under Tru64 UNIX:

- PCI/ISA Alpha 21164/366 MHz SBC with 2 MB cache and Tru64 UNIX license (EBM21-AZ)
- 512 MB main memory (2 x EBXMA-HC, for a total of four 128 MB DIMMs)
- Desktop enclosure with 14-slot PICMG backplane: 10 PCI, 3 ISA, 1 SBC (ETM33-BD)

- Eight PowerStorm 3D30 graphics cards (8 x SN-PBXGB-AA)
- PCI Ethernet card (DE450-CA)
- PCI fast/narrow SCSI controller (KZPAA-AA)
- Mandatory or associated options such as floppy drives, hard drives, CD-ROM drives, cable kit for PICMG enclosure, and power cord
- Tru64 UNIX Version 4.0E or higher
- DMCC SRM code from the Version 5.2 Firmware CD-ROM

1.1.5.2 Hardware Setup

When you configure the PCI/ISA 15-slot platform for 8-headed graphics, placement of the graphics cards is critical.

The qualified configuration (described in Section 1.1.5.1) uses an ETM33-BD desktop enclosure with a 14-slot backplane. Within that enclosure, the PCI option cards must be placed into PCI slots in top-to-bottom order as follows:

- Secondary 32-bit PCI bus connectors
  - KZPAA SCSI card
  - PowerStorm graphics card: SCREEN 2
  - PowerStorm graphics card: SCREEN 3
  - PowerStorm graphics card: SCREEN 4
  - DE450 Ethernet card
  - PowerStorm graphics card: SCREEN 5
  - PowerStorm graphics card: SCREEN 6
  - PowerStorm graphics card: SCREEN 7
- Primary 64-bit PCI bus connectors
  - PowerStorm graphics card: SCREEN 0 (VGA ENABLED)
  - PowerStorm graphics card: SCREEN 1

For reference, the power connector is situated above the PCI slots, and the SBC and ISA connectors are below.

All PowerStorm cards must have their Alias jumper IN and VGA EN jumper OUT, except the SCREEN 0 card, which must be VGA-enabled.

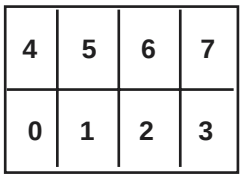
1.1.5.3 Software Setup

After you complete hardware configuration for the 8-headed system, you can set up the operating system to operate the eight screens as one row of eight screens (8x1) or two rows of four screens (4x2).

By default in a multiheaded configuration, the screens are operated as 8x1. To set up the screens in a 4x2 combination, you must edit your system's X Window System server configuration file, `/usr/var/X11/Xserver.conf`. Instructions for editing this file to customize the X server configuration are provided in the `Xserver`(1X) reference page.

To set up 4x2 operation, you need to specify `-edge_top`, `-edge_bottom`, `-edge_right`, and `-edge_left` command line arguments that arrange and attach the screens as you wish them. Each argument takes *scr1* and *scr2* values, which are the numbers of the screens you are attaching.

For example, you could arrange the eight screens as follows:



ZK-1559U-AI

To achieve this combination, add the appropriate command line arguments to the command line arguments section of `Xserver.conf`, as follows:

```
! you specify command line arguments here
args <
    -pn
    -edge_top0 4     -edge_top1 5     -edge_top2 6     -edge_top3 7
    -edge_bottom4 0  -edge_bottom5 1  -edge_bottom6 2  -edge_bottom7 3
    -edge_right0 1   -edge_right1 2   -edge_right2 3
    -edge_right4 5   -edge_right5 6   -edge_right6 7
    -edge_left1 0    -edge_left2 1    -edge_left3 2
    -edge_left5 4    -edge_left6 5    -edge_left7 6
>
```

1.1.6 Writing PCI Bus Device Drivers

For information about writing PCI bus device drivers, see the Tru64 UNIX Device Driver Kit (DDK), which is orderable separately from the base operating system.

You can browse a subset of device driver writing materials at the Tru64 UNIX Publications web site, currently located at the following URL:

http://www.tru64unix.compaq.com/docs/

Note

The Tru64 UNIX Publications web site also provides the latest DDK technical updates. DDK customers should check for potential DDK technical updates whenever they install a new version of the operating system.

1.2 Alpha VME 4/nnn and 5/nnn Single-Board Computers (EBVnn)

The Alpha VME 4/nnn and 5/nnn platforms are members of a family of VMEbus-based single-board computers (SBCs). The part numbers for these SBCs are EBV14-xx (Alpha VME 4/nnn) and EBV16-xx (Alpha VME 5/nnn).

Support for the VIP/VIC64 VMEbus adapter on the Alpha VME 4/nnn and 5/nnn SBCs is consistent with the support for this adapter on AXPvme SBCs and Alpha VME 2100 systems.

The following notes are specific to Alpha VME 4/nnn and 5/nnn SBCs.

1.2.1 Verifying CPU Version

You can use the `sizer` utility to identify the Alpha VME 4/nnn and 5/nnn SBCs. The `sizer -c` command displays the following output for Alpha VME 4/224 and 4/288 SBCs:

```
sysname> sizer -c
cpu      "DECALPHAVME_224"
```

The `sizer -c` command displays the following output for Alpha VME 5/352 and 5/480 SBCs:

```
sysname> sizer -c
cpu      "DECALPHAVME_320"
```

1.2.2 Firmware Requirements

Before installing the operating system, make sure that your system has the correct firmware version. The minimum firmware versions required are Version 1.2 or higher for an Alpha VME 4/224 or 4/288 SBC, and Version 1.0 or higher for an Alpha VME 5/352 or 5/480 SBC. If you have an earlier firmware version, update your firmware before installing the operating system software. For information on how to update your firmware, see the firmware documentation.

To determine the version of firmware on your system, enter the following command at the console firmware prompt:

```
>>> show version
```

1.2.3 Installing Tru64 UNIX

For information about installing the operating system on an Alpha VME 4/nnn or 5/nnn SBC, see the Tru64 UNIX *Installation Guide*. The *Installation Guide* provides platform-specific instructions for booting.

1.2.4 Configuring the VMEbus

For information about configuring the VMEbus for an Alpha VME SBC, see Chapter 2.

For information about setting up a VMEbus backplane–based network in which Alpha VME SBCs operate as Ethernet nodes, see Chapter 3.

1.2.5 Restrictions and Known Problems

The following restrictions apply to Alpha VME 4/nnn and 5/nnn SBCs.

1.2.5.1 VMEbus Autovectors Not Supported

The Alpha VME 4/nnn and 5/nnn SBCs do not support VMEbus autovectors.

1.2.5.2 Network Port Termination Required

An Alpha VME 4/nnn or 5/nnn SBC that has the network configured in an up state must have its external network connection properly terminated. If the network connection is unplugged or not properly terminated, then the network software will periodically time out and perform a network reset. This is normal for an unterminated Alpha VME system. However, it will cause high system latencies during the reset period, resulting in delays of about 10 milliseconds, which can affect the realtime performance of the system.

Note that a loopback connector is not sufficient to terminate the network connection.

1.2.5.3 Some TGA Video Card Configurations Require Disabling X Server DMA Writes

To use TGA video cards in some Alpha VME configurations, you must disable X server direct memory access (DMA) write operations. This restriction applies to the following configurations:

- EBVXG (TGA) video cards on Alpha 4/nnn and 5/nnn SBCs; note that the EBVXG video card is always installed behind the PCI-to-PCI bridge (PPB)

- TGA8 and TGA24 video cards on Alpha 5/nnn SBCs

The general procedure for disabling X server DMA write operations is as follows:

1. Bring the system to single-user mode.

   If you are able to use the `shutdown` command, execute the following command as superuser:

   ```
   # /usr/sbin/shutdown +2 "Disabling graphics DMA writes"
   ```

   If you cannot use the `shutdown` command (for example, if the X server on the video card is hung), you must halt your system by pressing the hardware halt button and then reboot your system to single-user mode by entering the following command:

   ```
   >>> boot -fl s
   ```

2. Mount all local file systems.

   After your system is in single-user mode, mount all of your local file systems by entering the following command:

   ```
   # bcheckrc
   ```

3. Change the directory to `/usr/var/X11` by entering the following command:

   ```
   # cd /usr/var/X11
   ```

4. Save a copy of the `Xserver.conf` file by entering a command such as the following:

   ```
   # cp Xserver.conf Xserver.conf.old
   ```

5. Edit the `Xserver.conf` file to add the text `-I -ffbDoDMA 4` to the command line arguments section. For example, if the command line arguments section is in its initial default state, it appears as follows:

   ```
   ! you specify command line arguments here
   args <
           -pn
   >
   ```

   Insert the text `-I -ffbDoDMA 4` after `-pn` as follows:

   ```
   ! you specify command line arguments here
   args <
           -pn -I -ffbDoDMA 4
   >
   ```

6. Return the system to multiuser mode by executing the following command:

```
# init 3
```

With this change, the video card and X server will run correctly on the SBC in multiuser mode.

1.2.5.4 Master Block Transfer Restrictions

For restrictions that apply to performing VMEbus master block transfers (MBLTs) using hardware DMA engines, see the discussion of MBLTs in Section 2.2.8 (VIP/VIC-based Alpha VME SBCs) or Section 2.3.11 (UNIVERSE II–based Alpha VME SBCs).

1.2.6 Writing VMEbus Device Drivers

For information about writing VMEbus device drivers, see the Tru64 UNIX Device Driver Kit (DDK), which is orderable separately from the base operating system.

You can browse a subset of device driver writing materials at the Tru64 UNIX Publications web site, currently located at the following URL:

http://www.tru64unix.compaq.com/docs/

Note

The Tru64 UNIX Publications web site also provides the latest DDK technical updates. DDK customers should check for potential DDK technical updates whenever they install a new version of the operating system.

1.3 AXPvme Single-Board Computers

The following notes are specific to the AXPvme single-board computers (SBCs). The part numbers for these SBCs include EBV10-xx (AXPvme 100) and EBV12-xx (AXPvme 166 and 230).

1.3.1 Firmware Upgrade Required

AXPvme SBCs must upgrade to Version 17.0 or higher of the AXPvme firmware to run the current version of the operating system.

1.3.2 Master Block Transfer Restrictions

The following restriction applies to the VIP/VIC adapter used on AXPvme SBCs and Alpha VME 2100 systems. Performing master block transfers (MBLTs) with a data width of D64 can produce unpredictable results in the following cases:

- If D64 slave access is performed before memory has been mapped to the VMEbus.
- If memory access does not coincide with the appropriate access mode, such as attempting user access to memory specified as supervisory-mode access.
- If the AXPvme SBC or Alpha VME 2100 system is a VMEbus interrupter and is targeted for D64 slave access. The interrupt vector presented by the VMEbus interrupter may not be the vector specified in the `vba_post_irq` routine.

Memory must be mapped to the VMEbus prior to D64 slave access. Access to memory must coincide with the appropriate access mode. If supervisory-mode access is specified when memory is mapped, memory accesses must use supervisory mode. If user-mode access is specified, both supervisory and user access are allowed.

See Section 2.2.7 and Section 2.2.8 for more information on slave and master block transfers, including additional restrictions that apply to MBLTs.

2

Configuring the VMEbus for Alpha VME Systems

This chapter explains how to configure the VMEbus for OEM platforms running Tru64 UNIX. The chapter provides an overview followed by sections that address groups of platforms based on their VMEbus adapter type:

- VMEbus support overview (Section 2.1)
- Configuring VIP/VIC-based Alpha VME SBCs (Section 2.2)
- Configuring UNIVERSE II–based Alpha VME SBCs (Section 2.3)

2.1 VMEbus Support Overview

The Tru64 UNIX operating system includes a generic VMEbus interface layer that provides customers with a consistent interface to VMEbus devices across Alpha workstation and server platforms and Alpha VME single-board computers (SBCs).

The operating system supports the following PCI/VMEbus adapters:

- UNIVERSE II PCI64-to-VME64 adapter
- VIP/VIC PCI32-to-VME64 adapter
- DWP64 PCI32-to-VME64 adapter
- DWPVC PCI32-to-VME32 adapter

Alpha VME SBCs provide an integrated PCI/VMEbus adapter: either VIP/VIC or UNIVERSE II. The DWP64 and DWPVC adapters are provided in layered product kits for use with Alpha workstation and server platforms.

For information about the VMEbus-based systems supported by the operating system, see the Tru64 UNIX Software Product Description (SPD).

This chapter provides information about configuring the VMEbus on the Alpha VME family of SBCs. To configure a VMEbus backplane (`vb`) network with Alpha VME SBCs in the same backplane communicating as network nodes, see Chapter 3.

To write VMEbus device drivers, you must obtain the Tru64 UNIX Device Driver Kit (DDK), which is available separately from the base operating

system. The DDK provides a detailed VMEbus device driver example that you can run on the Alpha VME SBCs.

To write VMEbus device drivers for Alpha workstation and server platforms with DWP64 or DWPVC adapters, you must have the associated adapter driver software and documentation in addition to the DDK. Be sure to check for the required processor and hardware configurations. For more information about the DWP64 and DWPVC adapters, see the PCI32/VME64 Adapter Driver SPD and the PCI/VME Adapter Driver SPD.

2.2 Configuring VIP/VIC-Based Alpha VME SBCs

This section describes how to set up VIP/VIC-based Alpha VME systems for use on the VMEbus, including how to modify attributes of the `vba_vipvic` kernel subsystem.

VMEbus setup allows you to run the operating system on the following VIP/VIC-based AXPvme and Alpha VME systems:

- AXPvme single-board computers (SBCs)
- Alpha VME 4/224 and 4/228 SBCs
- Alpha VME 5/352 and 5/480 SBCs
- Alpha VME 2100 system

For information about installing the operating system on the listed systems, see the *Installation Guide*.

For information about setting up UNIVERSE II–based Alpha VME systems for use on the VMEbus, see Section 2.3.

This section addresses the following topics relating to the use of the VMEbus on the listed systems:

- Configuring the `vba_vipvic` subsystem (Section 2.2.1)
- Configuring VMEbus A32 and A24 address spaces (Section 2.2.2)
- Configuring the VMEbus A16 address space (Section 2.2.3)
- Configuring VMEbus interrupts (Section 2.2.4)
- Using VMEbus hardware byte-swapping modes (Section 2.2.5)
- Sharing memory between big endian and little endian processors (Section 2.2.6)
- Performing VMEbus slave block transfers (Section 2.2.7)
- Performing VMEbus master block transfers with local DMA (Section 2.2.8)

- Using the realtime interrupt-handling routine `rt_post_callout` (Section 2.2.9)

2.2.1 Configuring the vba_vipvic Subsystem

This section describes how to configure the `vba_vipvic` kernel subsystem in order to prepare VIP/VIC-based AXPvme and Alpha VME systems for use on the VMEbus.

You configure the VIP/VIC adapter by examining the default (or current) attributes supplied for the `vba_vipvic` subsystem, determining which attributes (if any) you want to change, then modifying the `/etc/sysconfigtab` file on your machine. After modifying `/etc/sysconfigtab`, you must shut down and reboot the system.

Note

Do not directly edit `/etc/sysconfigtab`. Instead, use the `sysconfigdb` facility, as described in the `sysconfigdb`(8) reference page. It is recommended that you maintain private `sysconfigtab` file fragments for `vba_vipvic` attributes and use `sysconfigdb` switches to add (`-a -f`), delete (`-d`), or merge (`-m -f`) `vba_vipvic` attribute values. The example in Section 3.6 illustrates this approach. The `sys_attrs`(5) reference page provides additional guidelines for editing kernel subsystem attributes. You must always reboot after changing `vba_vipvic` subsystem attributes.

Common modifications to the `vba_vipvic` subsystem default attributes are to reconfigure the A32, A24, and A16 address spaces. For example, you could use `sysconfigdb` to edit the following modifications into `/etc/sysconfigtab`:

```
vba_vipvic:
        A32_Base = 0x10000000
        A32_Size = 0x08000000
        A24_Base = 0x00A00000
        A24_Size = 0x200000
        A16_Base = 0x00000000
```

In this example, the A24 inbound DMA window base address is modified from the default 0x00C00000 to 0x00A0000; the A24 window size from the default 4 MB to 2 MB; and the A16 interprocessor communication base address from the default 0x00000100 to 0x00000000.

You can modify values for the following VIP/VIC adapter attributes; each list item corresponds to a later subsection:

VMEbus request level (Section 2.2.1.1)
VIC arbitration mode (Section 2.2.1.2)
VMEbus fairness timer value (Section 2.2.1.3)
Local bus and VMEbus timeout periods (Section 2.2.1.4)
VMEbus release mode (Section 2.2.1.5)
System controller VMEbus resets (Section 2.2.1.6 and Section 2.2.1.7)
VIC master write posting (Section 2.2.1.8)
VMEbus DMA interleave gap (Section 2.2.1.9)
VMEbus DMA read limit (Section 2.2.1.10)
VMEbus DMA write limit (Section 2.2.1.11)
DMA method (hardware or emulated) for SMP systems (Section 2.2.1.12)

You can also modify the following values for the A32, A24, and A16 address spaces that the VMEbus hardware architecture defines; each list item corresponds to a later subsection:

A32 and A24 overlapping address configuration (Section 2.2.2.1)
A32 and A24 DMA inbound window sizes (Section 2.2.2.2)
A32 DMA inbound window base address (Section 2.2.2.3)
A24 DMA inbound window base address (Section 2.2.2.4)
A16 base for interprocessor communication facilities (Section 2.2.3)

Table 2–1 lists the defaults supplied for various VMEbus parameters. The default values specified should provide proper VMEbus operation for most applications. Be careful when modifying these values; not all adapters support all fields.

Table 2–1: VIP/VIC VMEbus Adapter Defaults

Parameter	Default	Meaning
VME_Br_Lev	0x03	Bus request level 3 for master cycles
VIC_Arb_Mode	0x00	Arbitration mode is round robin
VME_Fair_Req	0x00	VMEbus fair requester disabled
VIC_Loc_Bus_To	0x05	Local bus timeout period is 256 microseconds
VME_Bus_To	0x06	VMEbus timeout period is 512 microseconds
VIC_Rel_Mode	0	Release mode is release on request (ROR)
VIC_Syscon	1	System controller VMEbus reset is enabled
VIC_Wrt_Post	0	Disable VIC master write posting

Table 2–1: VIP/VIC VMEbus Adapter Defaults (cont.)

Parameter	Default	Meaning
VIC_DMA_Intrlv	15	DMA interleave gap is 3.75 microseconds (value * 250 nanoseconds)
Lmt_DMA_Rd	0	No DMA read limit
Lmt_DMA_Wrt	0	No DMA write limit
Frce_Hw_DMA	0	Do not force hardware DMA engine on SMP systems
A32_Base	0x08000000	A32 inbound DMA window base address
A32_Size	0x8000000	A32 window size (128 MB)
A24_Base	0x00C00000	A24 inbound DMA window base address
A24_Size	0x400000	A24 window size (4 MB)
A16_Base	0x00000100	A16 interprocessor communication base address
A16_Mask	0x00000000	A16 interprocessor communication mask
A24_A32_Ovrlap	1	Inbound A24/A32, if same space, overlap

Table 2–2 lists VMEbus interrupt parameters and their initial defaults. These defaults are later overwritten by system priority level (SPL) values supplied by the platform. See the SPL values listed in Table 2–3, or query the values at run time using the command `sysconfig -q vba_vipvic`.

Table 2–2: VIP/VIC VMEbus Interrupt Initial Defaults

Parameter	Default	Meaning
Irq0_SPL	3	VMEbus IRQ level to system SPL map
Irq1_SPL	3	VMEbus IRQ 1 to SPL SPLDEVLOW
Irq2_SPL	3	VMEbus IRQ 2 to SPL SPLDEVLOW
Irq3_SPL	3	VMEbus IRQ 3 to SPL SPLDEVLOW
Irq4_SPL	3	VMEbus IRQ 4 to SPL SPLDEVLOW
Irq5_SPL	3	VMEbus IRQ 5 to SPL SPLDEVLOW
Irq6_SPL	3	VMEbus IRQ 6 to SPL SPLDEVLOW
Irq7_SPL	3	VMEbus IRQ 7 to SPL SPLDEVLOW
Adapt_Blk_SPL	3	Adapter resource blocking SPL SPLDEVLOW
DMA_Access_Space	0	Adapter MBLT I/O access: sparse

2.2.1.1 Specifying the VMEbus Request Level

You can specify one of the following values for the VMEbus request level (parameter `VME_Br_lev`). The value is stored in the VIC64 Arbiter/Requester Configuration Register (ARCR).

0x00	VMEbus request level BR0
0x01	VMEbus request level BR1
0x02	VMEbus request level BR2
0x03	VMEbus request level BR3 (default)

2.2.1.2 Specifying the VIC Arbitration Mode

You can specify one of the following values for the VMEbus arbitration mode (parameter `VIC_Arb_Mode`). The VMEbus arbitration mode is stored in the VIC64 Arbiter/Requester Configuration Register (ARCR). This parameter is applicable only when the VMEbus adapter is configured to be the system controller.

0x00	VIC performs round-robin VMEbus arbitration (default)
0x01	VIC performs priority VMEbus arbitration

2.2.1.3 Specifying the VMEbus Fairness Timer Value

You can specify one of the following values for the Arbiter/Requester fair request timeout (parameter `VME_Fair_Req`). The fair request timeout value is stored in the VIC64 Arbiter/Requester Configuration Register (ARCR).

0x00	Fairness disabled (default)
0x01	Fair request timeout = 2 microseconds
0x02	Fair request timeout = 4 microseconds
0x03	Fair request timeout = 6 microseconds
0x04	Fair request timeout = 8 microseconds
0x05	Fair request timeout = 10 microseconds
0x06	Fair request timeout = 12 microseconds
0x07	Fair request timeout = 14 microseconds
0x08	Fair request timeout = 16 microseconds
0x09	Fair request timeout = 18 microseconds
0x0A	Fair request timeout = 20 microseconds

0x0B	Fair request timeout = 22 microseconds
0x0C	Fair request timeout = 24 microseconds
0x0D	Fair request timeout = 26 microseconds
0x0E	Fair request timeout = 28 microseconds
0x0F	Fair request timeout = none

2.2.1.4 Specifying Bus Timeout Periods

You can specify one of the following values for the local bus timeout period (parameter `VIC_Loc_Bus_To`) and for the VMEbus timeout period (parameter `VME_Bus_To`). Each value is stored in the VIC64 Transfer Timeout Register (TTR). The local bus timeout period must be shorter than the VMEbus timeout period.

0x00	Timeout = 4 microseconds
0x01	Timeout = 16 microseconds
0x02	Timeout = 32 microseconds
0x03	Timeout = 64 microseconds
0x04	Timeout = 128 microseconds
0x05	Timeout = 256 microseconds (default for local bus)
0x06	Timeout = 512 microseconds (default for VMEbus)
0x07	Timeouts disabled

2.2.1.5 Specifying the VMEbus Release Mode

You can specify one of the following values for the release mode (parameter `VIC_Rel_Mode`). The release-mode value is stored in the VIC64 Release Control Register (RCR).

0	Release on request (ROR) — the default
1	Release when done (RWD)

2.2.1.6 Specifying System Controller VMEbus Resets

You can specify one of the following values to indicate whether or not the adapter should issue VMEbus resets if it is the system controller (parameter `VIC_Syscon`).

For AXPvme SBCs and Alpha VME 4/nnn and 5/nnn SBCs, in addition to specifying a value from this list, you must set the configuration switches to indicate whether or not the SBC is the VMEbus system controller. See the

SBC's installation guide for information on setting the module configuration switches.

The Alpha VME 2100 adapter is always the VMEbus system controller. There are no module configuration switches to disable it from being the system controller.

The VMEbus backplane must have only one system controller. The system controller must be electrically the first module in the VMEbus backplane and in most systems must be in the first VMEbus slot.

0	Do not issue VMEbus resets if system controller
1	Issue VMEbus resets if system controller (default)

The values specified interact with the VMEbus initialization code to determine whether a VMEbus reset is issued when the VMEbus adapter is being configured. If the value is set to 1 and the system being booted is the system controller, as determined by the VMEbus initialization code, a VMEbus reset is issued. If you do not want a VMEbus reset issued during VMEbus adapter configuration, set the value to 0 (zero). These values pertain only to the system controller.

If the system controller is configured to issue a VMEbus reset during adapter initialization, and other processor modules are installed in the VMEbus backplane, boot the system controller first to allow devices and processor modules to perform their bus reset actions.

2.2.1.7 Special Considerations for VMEbus Resets

The system controller should always be the initiator of VMEbus resets. However, under certain error conditions, other VMEbus adapter modules may invoke a VMEbus reset. Modules installed in the VMEbus backplane react to bus resets differently. Some modules, if configured, perform a module reset. Some may have their VMEbus interface reset to a power-up state without notification to the operating system. This could leave the VMEbus adapters in an unconfigured state, cause unwanted effects to the operating system and its device drivers, and cause VMEbus errors to occur. Other VMEbus adapters on the VMEbus may accept VMEbus resets and attempt to reconfigure themselves to the hardware context they were running before the bus reset occurred. However, device drivers expecting interrupts may not receive them and I/O hardware operations may be canceled by the VMEbus reset without notification to the device driver. There is also a potential for data corruption to occur when the VMEbus adapter is reset during an I/O operation.

It is recommended that the system controller be the initiator of VMEbus resets during adapter initialization. If the system controller is not controlled

by a processor, then a power-up sequence should cause all VMEbus adapters and devices to be reset. All modules on the VMEbus should perform a module reset upon detection of a bus reset. VMEbus adapters that are not the system controller and that are running an operating system should be shut down in an orderly fashion prior to the system controller being booted. These VMEbus adapters should be rebooted after the system controller has been booted, providing that the system controller is to be used and controlled by a processor.

For Alpha VME 2100 systems, the VMEbus adapter can be the initiator of VMEbus resets only. Upon receipt of a bus reset, its VMEbus interface (VIC64) is reset. The reset state of the VMEbus interface (VIC64) is not the VMEbus adapter's configured state. The operating system and device drivers are not notified that a bus reset has occurred. If the adapter is accessed or if an I/O operation is invoked following a bus reset, the access may result in an adapter error, misoperation, or system crash.

For AXPvme SBCs and Alpha VME 4/nnn and 5/nnn SBCs, it is recommended that nodes that are not the system controller have their module configuration switch 3 set to Closed (resets the SBC module in response to a VMEbus reset signal). When the VMEbus is reset, and the module switch is set to accept a VMEbus reset, nonsystem controller modules take a boot action and are reset to a powered state.

If the SBC module configuration switch 3 is set to Open (does not reset the SBC module in response to a VMEbus reset signal), the VMEbus adapter software will receive a VMEbus reset interrupt upon detection of a bus reset. The VMEbus reset signal initializes the VMEbus adapter (VIC64) to its power-up state. The VMEbus reset interrupt service interface displays the following message on the console terminal:

```
vba0 reset_inter:  VMEbus reset detected
```

The interrupt service interface then initializes the VMEbus adapter to its defaults and enables any previously enabled interrupt enable bits.

Do not set the SBC module configuration switch 3 to Open without considering the following side effects of receiving a VMEbus reset: device drivers expecting interrupts may not receive them and I/O hardware operations may be canceled by the VMEbus reset without notification to the device drivers. There is potential risk of data corruption depending upon I/O activity at the time a bus reset occurred.

2.2.1.8 Specifying VMEbus Master Write Posting

Master write posting is not currently supported. Do not change the value from the default of 0 (zero) or unpredictable results may occur.

2.2.1.9 Specifying the VMEbus DMA Interleave Gap

You can specify one of the following values for the DMA interleave gap (parameter `VIC_DMA_Intrlv`), which is the time period between master block transfer (MBLT) DMA bursts. The DMA interleave gap value is stored in the VIC64 Block Transfer Control Register (BTCR) at the start of a master block transfer DMA. This parameter is applicable only when you use the VMEbus adapter's hardware DMA engine to perform the DMA.

15	Interleave gap = 3.75 microseconds (default)
14	Interleave gap = 3.50 microseconds
13	Interleave gap = 3.25 microseconds
12	Interleave gap = 3.00 microseconds
11	Interleave gap = 2.75 microseconds
10	Interleave gap = 2.50 microseconds
9	Interleave gap = 2.25 microseconds
8	Interleave gap = 2.00 microseconds
7	Interleave gap = 1.75 microseconds
6	Interleave gap = 1.50 microseconds
5	Interleave gap = 1.25 microseconds
4	Interleave gap = 1.00 microseconds
3	Interleave gap = 0.75 microseconds
2	Interleave gap = 0.50 microseconds
1	Interleave gap = 0.25 microseconds
0	Interleave gap = 0.00 microseconds

Caution

You must not specify the value 0 (zero) if D64 master block transfers are to be performed. Unpredictable errors and possible data corruption may result if you specify 0 (zero) with D64 transfers.

During the DMA interleave gap, stalled or new programmed I/O (PIO), VMEbus IACK cycles, or slave DMAs may obtain the bus to perform the required I/O operation. The VIC64 is enabled for dual paths to allow these I/O operations to occur during the DMA interleave gap. Changing this parameter arbitrarily may cause unwanted side effects.

Decreasing the value from the default increases DMA throughput. However, as the number approaches 0 (zero), outstanding PIO operations, VMEbus IACKs, and slave DMAs may be held off from obtaining the bus until the DMA in progress is completed. These operations might have occurred during the DMA interleave gaps if the default value had been used.

Specifying a small DMA interleave gap may result in PCI retry timeouts, poor PIO performance, increased interrupt response time, other PCI transactions being held off, and possible system time loss. Beware of these side effects when specifying a new value for the DMA interleave gap.

2.2.1.10 Specifying Limits on VMEbus DMA Reads

You can specify one of the following values to enable or disable an 8 KB limit on DMA read operations (parameter `Lmt_DMA_Rd`). Placing an 8 KB limit on DMA reads can enhance throughput when the bus is busy. Transfers relinquish the bus after each 8 KB or less.

0	No DMA read limit (default)
1	Limit DMA reads to 8 KB or less

2.2.1.11 Specifying Limits on VMEbus DMA Writes

You can specify one of the following values to enable or disable an 8 KB limit on DMA write operations (parameter `Lmt_DMA_Wrt`). Placing an 8 KB limit on DMA writes can enhance throughput when the bus is busy. Transfers relinquish the bus after each 8 KB or less.

0	No DMA write limit (default)
1	Limit DMA writes to 8 KB or less

2.2.1.12 Specifying the DMA Method for SMP

You can specify one of the following values to enable or disable use of the hardware DMA engine on an SMP system (parameter `Frce_Hw_DMA`). Note that in an SMP system, you would enable use of the hardware DMA engine only if the system was known to be quiescent, with no other PIO, DMA, or interrupt activity occurring on the bus.

0	Use emulated DMA on SMP system (default)
1	Force hardware MBLT on SMP system

2.2.2 Configuring VMEbus A32 and A24 Address Spaces

As part of configuring the `vba_vipvic` kernel subsystem, you can configure the VMEbus 32-bit address space (A32) and 24-bit address space (A24) for your system. A32 and A24 space are used for direct memory access (DMA) inbound windows.

The A32 space has a maximum size of 4 GB and can be partitioned into 32 128 MB windows. You can further partition each 128 MB window in increments as small as 16 MB. Valid window segments are 16, 32, and 64 MB.

The A24 space has a maximum size of 16 MB, and the base address is always zero (0x00000000). This means that you can partition the address space but cannot move it. The default window size is 4 MB and the base address for a window must be a multiple of the window size. The default inbound window is the top 4 MB of the 16 MB space.

You can specify whether the A24 and A32 addresses can reside within the same addressing range or whether they must be unique.

2.2.2.1 Specifying A32 and A24 Address Space Overlapping

Read this section if the A32 direct memory access (DMA) inbound window will be configured with a base address of zero (0x00000000), overlapping the A24 address space. A24 inbound windows are always configured within the first 16 MB of the 4 GB VMEbus address space.

Typically, VMEbus A24 and A32 address spaces overlap each other such that addresses in each address space are unique to that address space. As an example, address 0x200 in A32 address space is not the same address as 0x200 in A24 address space. This is the default configuration, selected if you leave the `A24_A32_Ovrlap` parameter at its default value of 1.

You can configure some VMEbus devices to recognize the same VMEbus address in both A24 and A32 address spaces. These devices treat the two

address spaces as a single entity. Consult the VMEbus hardware device manuals to determine if any devices installed on the VMEbus follow this model. If so, you must configure the autoconfiguration software to disallow A32 DMA allocations within the first 16 MB of VMEbus address space. If you do not do this, an A32 direct memory access to the first 16 MB of VMEbus address space by another VMEbus device may not only select the AXPvme or Alpha VME module but also select the device that treats the address spaces as a single entity.

Configure the first 16 MB of VMEbus address space as a single entity by setting the `A24_A32_Overlap` parameter to 0 (zero).

The values for overlapping and unique address spaces are as follows. These values are valid only when the A32 and A24 address spaces are configured to overlap each other; that is, when the A32 base address equals zero (0x00000000).

0	A24 and A32 addresses must be unique
1	A24 and A32 addresses can overlap each other (default)

2.2.2.2 Configuring A32 and A24 Window Sizes

You can specify the DMA inbound window size for the A32 address space (parameter `A32_Size`) and the A24 address space (parameter `A24_Size`).

If you specify an invalid base address in relation to a window size, the autoconfiguration code adjusts the base address to match the window size. The base address is adjusted downward to the next appropriate boundary for the window size.

The window size values are as follows:

0x10000	64 KB
0x20000	128 KB
0x40000	256 KB
0x80000	512 KB
0x100000	1024 KB (1 MB)
0x200000	2048 KB (2 MB)
0x400000	4096 KB (4 MB) [A24 default]
0x800000	8192 KB (8 MB)
0x1000000	16384 KB (16 MB)
0x2000000	32768 KB (32 MB)

0x4000000	65536 KB (64 MB)
0x8000000	131072 KB (128 MB) [A32 default]

2.2.2.3 Specifying the A32 Base Address

You specify the A32 base address using the A32_Base parameter. The following table lists the values used for partitioning 128 MB windows in the A32 address space. Note that the base value is contained in bits 24 through 31, with bits 27 through 31 indicating the window and bits 24 through 26 indicating the partition size.

Base (Bits 31-24)	Bus Address (A32_Base)	Bus Offset
128 MB window:		
0000 0000	0x00000000	0 MB
64 MB windows:		
0000 0000	0x00000000	0 MB
0000 0100	0x04000000	64 MB
32 MB windows:		
0000 0000	0x00000000	0 MB
0000 0010	0x02000000	32 MB
0000 0100	0x04000000	64 MB
0000 0110	0x06000000	96 MB
16 MB windows:		
0000 0000	0x00000000	0 MB
0000 0001	0x01000000	16 MB
0000 0010	0x02000000	32 MB
0000 0011	0x03000000	48 MB
0000 0100	0x04000000	64 MB
0000 0101	0x05000000	80 MB
0000 0110	0x06000000	96 MB
0000 0111	0x07000000	112 MB

2.2.2.4 Specifying the A24 Base Address

You specify the A24 base address using the A24_Base parameter. The following table lists the base address values for windows in the A24 address

space. The base address is stored in bits 16 through 23. The table has been truncated for the smaller window sizes.

Base (Bits 23-16)	Bus Address (A24_Base)	Bus Offset
16 MB window:		
0000 0000	0x00000000	0 MB
8 MB windows:		
0000 0000	0x00000000	0 MB
1000 0000	0x00800000	16 MB
4 MB windows:		
0000 0000	0x00000000	0 MB
0100 0000	0x00400000	4 MB
1000 0000	0x00800000	8 MB
1100 0000	0x00C00000	12 MB
2 MB windows:		
0000 0000	0x00000000	0 MB
0010 0000	0x00200000	2 MB
0100 0000	0x00400000	4 MB
0110 0000	0x00600000	6 MB
1000 0000	0x00800000	8 MB
1010 0000	0x00A00000	10 MB
1100 0000	0x00C00000	12 MB
1110 0000	0x00E00000	14 MB
1 MB windows:		
0000 0000	0x00000000	0 MB
0001 0000	0x00100000	1 MB
0010 0000	0x00200000	2 MB
0011 0000	0x00300000	3 MB
0100 0000	0x00400000	4 MB
0101 0000	0x00500000	5 MB
0110 0000	0x00600000	6 MB
0111 0000	0x00700000	7 MB
1000 0000	0x00800000	8 MB

Base (Bits 23-16)	Bus Address (A24_Base)	Bus Offset
1001 0000	0x00900000	9 MB
1010 0000	0x00A00000	10 MB
1011 0000	0x00B00000	11 MB
1100 0000	0x00C00000	12 MB
1101 0000	0x00D00000	13 MB
1110 0000	0x00E00000	14 MB
1111 0000	0x00F00000	15 MB
512 KB windows:		
0000 0000	0x00000000	0 KB
0000 1000	0x00080000	512 KB
0001 0000	0x00100000	1024 KB
0001 1000	0x00180000	1536 KB
0010 0000	0x00200000	2048 KB
...	...	...
1111 1000	0x00F80000	15872 KB
256 KB windows:		
0000 0000	0x00000000	0 KB
0000 0100	0x00040000	256 KB
0000 1000	0x00080000	512 KB
0000 1100	0x000C0000	786 KB
0001 0000	0x00100000	1024 KB
...	...	...
1111 1100	0x00FC0000	16128 KB
128 KB windows:		
0000 0000	0x00000000	0 KB
0000 0010	0x00020000	128 KB
0000 0100	0x00040000	256 KB
0000 0110	0x00060000	384 KB
0000 1000	0x00080000	512 KB
...	...	...
1111 1110	0x00FE0000	16256 KB

Base (Bits 23-16)	Bus Address (A24_Base)	Bus Offset
64 KB windows:		
0000 0000	0x00000000	0 KB
0000 0001	0x00010000	64 KB
0000 0010	0x00020000	128 KB
0000 0011	0x00030000	192 KB
0000 0100	0x00040000	256 KB
...	...	...
1111 1111	0x00FF0000	16320 KB

2.2.3 Configuring the VMEbus A16 Address Space

As part of configuring the `vba_vipvic` kernel subsystem, you can configure the VMEbus 16-bit address space (A16) for your system. A16 space is used for interprocessor communication and to communicate with A16 VMEbus devices.

The A16 space has a maximum size of 64 KB and runs from VMEbus address 0000 hexadecimal to FFFF hexadecimal. You can configure the VMEbus Interprocessor Communication Facilities (ICF) of the AXPvme SBC, Alpha VME 4/nnn or 5/nnn SBC, or Alpha VME 2100 system on any 256-byte boundary within the VMEbus A16 address space. The default base address (parameter `A16_Base`) is 0x00000100. The mask value (parameter `A16_Mask`) must be left at zero (0x00000000).

2.2.4 Configuring VMEbus Interrupts

This section addresses VMEbus interrupt request levels and how to configure VMEbus interrupts in the software.

2.2.4.1 VMEbus Interrupt Request Levels

Table 2–3 lists the system priority levels (SPLs) at which VMEbus and VMEbus adapter interrupt requests are delivered to the operating system and device drivers. You can query your system's VMEbus SPLs at run time by issuing the command `sysconfig -q vba_vipvic`.

Table 2–3: VIP/VIC VMEbus Interrupt Request Levels

Interrupt Request Name	AXPvme SBC SPLs	Alpha VME SBC SPLs	Alpha VME 2100 SPLs
VMEbus IRQ 1	SPLDEVLOW	SPLDEVLOW	SPLDEVLOW
VMEbus IRQ 2	SPLDEVLOW	SPLDEVLOW	SPLDEVLOW
VMEbus IRQ 3	SPLDEVLOW	SPLDEVLOW	SPLDEVLOW
VMEbus IRQ 4	SPLDEVHIGH	SPLDEVHIGH	SPLDEVLOW
VMEbus IRQ 5	SPLDEVHIGH	SPLDEVHIGH	SPLDEVLOW
VMEbus IRQ 6	SPLDEVHIGH	SPLDEVHIGH	SPLDEVLOW
VMEbus IRQ 7	SPLDEVRT	SPLDEVRT	SPLDEVLOW
Autovector IRQ 1	SPLDEVLOW		
Autovector IRQ 2	SPLDEVLOW		
Autovector IRQ 3	SPLDEVLOW		
Autovector IRQ 4	SPLDEVHIGH		
Autovector IRQ 5	SPLDEVHIGH		
Autovector IRQ 6	SPLDEVHIGH		
Autovector IRQ 7	SPLDEVRT		
VMEbus Reset	SPLDEVRT	SPLDEVRT	
Module Switches	SPLDEVRT	SPLDEVRT	SPLDEVLOW
VMEbus IACK	SPLDEVLOW	SPLDEVLOW	SPLDEVLOW
DMA Status	SPLDEVRT	SPLDEVRT	SPLDEVLOW

The Alpha VME 4/nnn and 5/nnn SBCs do not support autovector requests. The Alpha VME 2100 system does not support autovector or VMEbus reset interrupt requests.

As Table 2–3 indicates, AXPvme and Alpha VME SBCs generate interrupt requests that higher-level interrupt requests can preempt, while Alpha VME 2100 interrupt requests are all delivered at the same SPL and cannot be preempted.

On the AXPvme and Alpha VME SBCs, device drivers must use the `rt_post_callout` routine for interrupts delivered at SPLDEVRT. Interrupt requests for which this is needed are VMEbus IRQ7, Autovector IRQ7, and any of the four module switch interrupts. Device drivers written for the SBCs that use the `rt_post_callout` routine will also run on the Alpha VME 2100 system without modifications.

Note

VMEbus device drivers written for Alpha VME 2100 systems, or for other platforms that deliver VMEbus interrupts at a single SPL, may be affected when run on the AXPvme or Alpha VME SBC platforms. If these device drivers are using SPLs to protect common resources between thread and interrupt service interfaces, the preempted interrupts of the SBC systems may have unwanted effects on the drivers. If these device drivers are servicing interrupts for VMEbus IRQ7, Autovector IRQ7, or module switch interrupts, then the drivers must be modified to use the `rt_post_callout` routine. Device drivers cannot invoke normal thread wakeup mechanisms at SPLDEVRT.

2.2.4.2 Setting VMEbus Interrupt Vector Parameters

You specify vectors and interrupt requests (IRQs) for a device driver using the `Vector` and `Bus_Priority` fields of a `VBA_Option` entry in the `/etc/sysconfigtab` file or in a `sysconfigtab` file fragment.

Device drivers are passed this information in the `controller` structure elements `ivnum` and `bus_priority`.

VMEbus interrupt vectors 24 to 255 are available to device drivers. Vectors 0 to 23 are reserved by the VMEbus adapter. When you specify a vector to the `Vector` field of `VBA_Option`, you must also use the `Bus_Priority` field to specify an IRQ. Valid IRQ specifications are values 1 through 7. These values correspond to VMEbus levels IRQ1 through IRQ7.

Note that if a VMEbus device uses an IRQ, that same IRQ cannot be used for autovectored interrupts.

See the Autoconfiguration Support section of *Writing VMEbus Device Drivers* (available in the Device Driver Kit) for an example of adding and enabling VMEbus interrupts. See the `vme_handler_info` structure in *Writing VMEbus Device Drivers* for interrupt handler information.

2.2.4.3 Specifying Autovector Interrupt Vectors

The Alpha VME 4/nnn, 5/nnn, and 2100 platforms do not support autovectors.

VMEbus devices of the type Release of Register Access (RORA) use autovectors. RORA devices are incapable of presenting a status/ID vector in the manner of Release On Acknowledge (ROAK) VMEbus devices.

RORA devices present an interrupt request to the system at a specified VMEbus IRQ level. Upon receipt of the interrupt request, the system

provides a system-defined status/ID vector and dispatches it to the interrupt service interface installed for the autovector. The device driver is responsible for dismissing the RORA device's interrupt request by performing a read or write access to the device. See the hardware manual for the RORA device to determine what type of access is needed to dismiss the interrupt request.

To select an autovector, use the `Vector` and `Bus_Priority` fields of `VBA_Option`. Specify a vector value of 0 (zero) and an IRQ value of 1 through 7, corresponding to VMEbus levels IRQ1 through IRQ7.

If an IRQ is used for an autovector, the same IRQ cannot be used for VMEbus interrupt vectors.

2.2.4.4 Specifying Module Switch Interrupt Vectors

Specify one of the following vectors in the `Vector` field of `VBA_Option` to select the module switch interrupt you want. Use the `Bus_Priority` field to specify 7 as the IRQ level.

Module switch 0 Vector 0x1140 [A16 offset 0x21]

Module switch 1 Vector 0x1150 [A16 offset 0x23] (default)

Module switch 2 Vector 0x1160 [A16 offset 0x25]

Module switch 3 Vector 0x1170 [A16 offset 0x27]

Module switch interrupt vectors allow a module to issue an interrupt to itself or to another module. The autoconfiguration software provides control and status registers (CSRs) for use in module switch interrupts. You can specify two CSRs in a `VBA_Option` entry in the `/etc/sysconfigtab` file or in a `sysconfigtab` file fragment. At boot time, the system searches for the specified CSRs.

The autoconfiguration software performs the appropriate bus mapping and provides `io_handle_t` values in the `addr` and `addr2` members of the driver's `controller` structure. The `addr` argument is passed to the driver's `probe` routine, while the `addr2` value must be obtained from the `addr2` member of the `controller` structure.

For example, the following `VBA_Option` entry specifies a CSR for the base address of the A16 Interprocessor Communication Facilities (ICF). The module switch 1 CSR is an offset from this A16 address.

```
VBA_Option = Csr1 - 0x100, ..., Vector - 0x1150, Bus_Priority - 7, ...
```

The `driver` structure allows you to specify the size, address type, and swap mode for the CSRs. For example, the following members in a `driver` structure indicate that the first CSR has a size of 256 bytes, is in the A16 address space, and is set to noswap mode:

```
int     addr1_size      256
int     addr1_atype     VME_A16_SUPER_ACC | VME_BS_NOSWAP
```

For more information, see the Device Driver Kit manuals *Writing Device Drivers* and *Writing VMEbus Device Drivers*, especially the sections on the `addr` and `addr2` members of the `controller` structure and on the `addr1_size`, `addr1_atype`, `addr2_size`, and `addr2_atype` members of the `driver` structure.

In addition, you can use the `vba_map_csr` routine to provide module switch interrupts. After using the `vba_map_csr` routine to create an I/O handle, you write to an address derived from the base address plus an offset. Two write operations are performed, one signifying a clear and one a set. The following code fragment shows how the I/O handle is created:

```
io_handle_t     ioh;           /* Define type of ioh */
vme_addr_t      A16base=0x100; /* Base CSR address */
ioh = vba_map_csr(ctlr, A16base, 256,
                  (VME_A16_SUPER_ACC |
                   VME_BS_NOSWAP));
```

The following code fragment shows how the module switch interrupts are issued:

```
write_io_port(ioh+0x22, 1, 0, 0)  /* Write to A16 base address
                                     plus the offset to clear
                                     module switch 1 */
mb();
write_io_port(ioh+0x23, 1, 0, 0)  /* Write to A16 base address
                                     plus the offset to set
                                     module switch 1 */
mb();
```

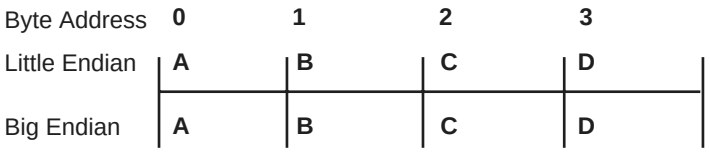
2.2.4.5 Specifying Global Switch Interrupt Vectors

Global switch interrupts are not currently supported.

2.2.5 Using VMEbus Hardware Byte-Swapping Modes

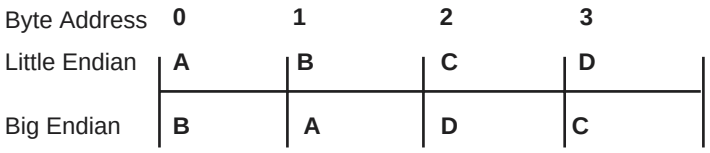
Alpha processors are little endian, while the VMEbus is big endian. The default byte-swapping mode, VME_BS_NOSWAP, causes the transfer of bytes between Alpha processors and the VMEbus to be arranged correctly. If, however, a 16-bit or 32-bit number is needed in a VMEbus register, the VME_BS_NOSWAP mode rearranges the bytes within the transfer such that the bytes are reversed in significance. Two other modes are provided to handle these situations: VME_BS_BYTE and VME_BS_LWORD. A third mode for swapping words within longwords, VME_BS_WORD, is not portable across VMEbus adapters and is provided for convenience. The definitions for these modes are in the `io/dec/vme/vbareg.h` file. The flags for these modes are used in `vba_map_csr`, in `dma_map_alloc` or `dma_map_load`, and in the `driver` structure.

VME_BS_NOSWAP mode provides a hardware mechanism for data coherency for byte-data transfers from Alpha processors (little endian) to the VMEbus (big endian). The address of any byte as seen on the two buses remains the same. Block transfers of byte information use 16- or 32-bit transfers. The transfer sizes are 8-, 16-, or 32-bits of byte information. Noswap-mode byte addressing is as follows:



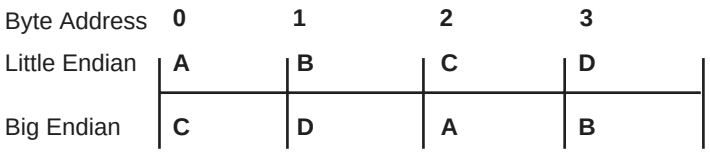
ZK-1560U-AI

VME_BS_BYTE mode provides a hardware mechanism for data coherency for 16-bit data transfers across the VMEbus, such as loading a 16-bit counter on a VMEbus device. In this mode, bytes within words are swapped. For portability, use only 16-bit aligned transfers. Byte swap-mode byte addressing is as follows:



ZK-1561U-AI

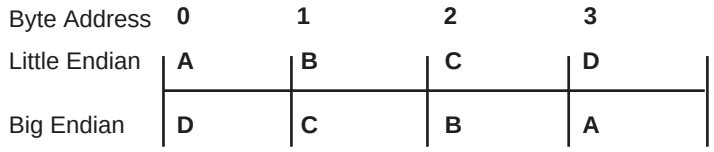
VME_BS_WORD mode provides a hardware mechanism for swapping words within longwords on certain VMEbus adapters. This mode is not portable across VMEbus adapters; on other VMEbus adapters, byte swapping may be dependent on data size. For AXPvme and Alpha VME platforms, system word swap-mode byte addressing is as follows:



ZK-1562U-AI

VME_BS_LWORD mode provides a hardware mechanism for data coherency for 32-bit data transfers across the VMEbus, such as loading a 32-bit

VMEbus address register. In this mode, bytes within words are swapped and words within longwords are swapped. The transfer size is 32 bits only. For portability, use only 32-bit transfers. Longword swap-mode byte addressing is as follows:



ZK-1563U-AI

2.2.6 Sharing Memory Between Big Endian and Little Endian Processors

In a shared memory environment, where packed data structures in common memory are shared between an Alpha processor (little endian) and a big endian processor, software byte swapping is required to arrange bytes properly for 16- or 32-bit quantities (such as 16-bit counter values or 32-bit VMEbus address values).

The following combination is recommended: VME_BS_NOSWAP with software byte swapping on nonbyte data for the Alpha processor; and no swapping on the big endian processor.

You could implement software swapping with read/write macros that perform the swap with the following code. The purpose here is to provide code that would run on both little endian and big endian machines that have shared memory.

```
#define read_word/long(iohandle,data)                    /
        data = read_io_port(iohandle,sizeof(word/long),0);/
#ifdef LITTLEENDIAN                                      /
        swap_xx(data);                                   /
#else /* BIGENDIAN */                                   /
#endif
#define write_word/long(iohandle,data)                   /
#ifdef LITTLEENDIAN                                      /
        swap_xx(data);                                   /
#else /* BIGENDIAN */                                   /
        write_io_port(iohandle,sizeof(word/long),0,data); /
#endif
```

2.2.7 Performing VMEbus Slave Block Transfers

The AXPvme and Alpha VME platforms are configured during adapter initialization to accept slave block transfers (SBLTs) with data widths of D16, D32, or D64. After the SBC has mapped its memory onto the VMEbus by using the `dma_map_alloc` and `dma_map_load` routines, no other user

interaction is needed. For information on calling the `dma_map_alloc` and `dma_map_load` routines, see the corresponding reference pages in the Device Driver Kit (available separately from the base operating system).

Memory must be mapped to the VMEbus prior to D64 slave access.

Access to memory must coincide with the appropriate access mode. If you specify supervisory-mode access when memory is mapped, memory accesses must use supervisory mode. If you specify user-mode access, both supervisory and user access are allowed.

2.2.8 Performing VMEbus Master Block Transfers with Local DMA

The VMEbus interfaces for the AXPvme and Alpha VME platforms provide a block-mode DMA engine. This DMA engine is capable of transferring up to 64 KB of data without processor intervention, in VMEbus data widths of D16, D32, or D64.

The DMA engine transfers data from the VMEbus to system memory (read) or from system memory to the VMEbus (write). The hardware interface handles the segmentation of the transfer. This ensures that the VMEbus specification is not violated in relation to crossing VMEbus 256-byte boundaries for D16 and D32 or 2-KB boundaries for D64.

The DMA engine is configured to give up the VMEbus during the transfer and to rearbitrate for the VMEbus again to continue the DMA. The time between when the DMA engine gives up the bus and rearbitrates for the bus is called the **interleave** period. During the interleave period, single-cycle VMEbus cycles, receipt of slave block transfers (SBLTs), or other operations may be performed.

The master block transfer (MBLT) hardware interface presents address modifiers of user block or supervisory block to the VMEbus, based on parameters passed in the software programming interface. The device or system on the VMEbus must be able to interpret these address modifiers; otherwise, bus errors may occur.

You can use the MBLT hardware interface to:

- Transfer data to and from those VMEbus devices that do not have their own DMA engine
- Move data between VMEbus device memory and system memory
- Transfer data to and from other systems that have their memory mapped to the VMEbus

The MBLT hardware interface supports DMA block-mode transfers to and from VMEbus A24 and A32 address space only.

2.2.8.1 Routines for Master Block-Mode Transfers

To use master block transfers (MBLTs) with the local hardware DMA engine, you must invoke the following routines and supply specific flag values:

```
vba_set_dma_addr
dma_map_alloc
dma_map_load
vba_dma
dma_map_unload
dma_map_dealloc
```

For information on calling these routines, see the corresponding reference pages in the Device Driver Kit (available separately from the base operating system).

The flag values DMA_IN and DMA_OUT have specific meaning for VMEbus support with respect to the `dma_map_alloc`, `dma_map_load`, and `vba_dma` routines. These flags indicate to the low-level VMEbus `dma_map_alloc`, `dma_map_load`, and `vba_dma` routines that the MBLT hardware DMA engine is to be used and the direction of the transfer.

Specifying DMA_IN implies a read from the VMEbus to system memory. Specifying DMA_OUT implies a write from system memory to the VMEbus. You use the `vba_set_dma_addr` routine to pass the flag values and the VMEbus address at which the transfer is to occur.

The VMEbus block-mode DMA engine on the VMEbus adapter is a single entity that must be shared among various device drivers. Specifying DMA_SLEEP causes the device driver to block in the `vba_dma` routine if the DMA engine is already being used. If DMA_SLEEP is not specified and the DMA engine is being used, `vba_dma` returns an error.

The following sample code shows how to invoke the MBLT hardware DMA engine for a block-mode read operation. The code uses a VMEbus transfer width of D32 to invoke a 256 KB transfer from VMEbus address A24 0x400000 to system memory. The code also allocates resources to handle transfers up to 1 MB in size. This allows `dma_map_load` and `vba_dma` to be invoked multiple times with varying size buffers. You can change the code to perform writes by substituting DMA_OUT for DMA_IN.

```
struct controller *ctlr;
vme_addr_t        vme_addr = 0x40000;
unsigned long     max_bc = (1024*1024);
unsigned long     rtn_bc;
char              *buffer;
unsigned long     buffer_bc = (1024 * 256);
sglist_t          dma_handle = (sglist_t)NULL;
vme_atype_t       flags = (VME_A24_UDATA_D32|DMA_IN|DMA_SLEEP);
int               rtn_flags;
/*
 *  Allocate a buffer (256 KB) to be used for the transfer
```

```
 */
MALLOC(buffer,(char *),buffer_bc,M_DEVBUF,M_WAITOK);
/*
 *  Specify a VMEbus address of 0x40000
 *  Specify flags
 *     A24 address space
 *     User mode
 *     Select DMA engine for a read (DMA_IN) and
 *       wait for DMA engine (DMA_SLEEP)
 */
rtn_flags = (int)vba_set_dma_addr(ctlr,flags,vme_addr);
/*
 *  Allocate DMA resources for up to 1 Mbyte transfer
 *    Specify flags returned from vba_set_dma_addr() above
 *    The return value from dma_map_alloc() should equal max_bc
 */
rtn_bc = dma_map_alloc(max_bc,ctlr,&dma_handle,rtn_flags);
/*
 *  Call dma_map_load() to load the resources for the
 *  DMA block-mode engine
 *    Specify the dma_handle returned from dma_map_alloc()
 *    Specify flags returned from vba_set_dma_addr()
 *    The return value from dma_map_load() should equal buffer_bc
 */
rtn_bc = dma_map_load(buffer_bc,
                      (vm_offset_t)buffer,
                      0,
                      ctlr,
                      &dma_handle,
                      0,
                      rtn_flags);
/*
 * Call vba_dma() to start up and monitor the VME adapter's block-mode
 *   DMA engine. Specify the dma_handle returned from dma_map_alloc.
 *   The return value from vba_dma() is the actual bytes transferred.
 *   This value should be the same as value buffer_bc. If not, then
 *   an error was detected during the transfer.
 */
rtn_bc = vba_dma(ctlr,dma_handle);
/*
 *  Unload and free DMA resources
 */
dma_map_unload(0,dma_handle)
dma_map_dealloc(dma_handle)
```

2.2.8.2 Restrictions on VMEbus Master Block Transfers

The following restrictions apply to using master block transfers (MBLTs) on the Alpha VME and AXPvme platforms. Failure to adhere to these restrictions may result in data loss during DMA transfers. These restrictions are listed by DMA transfer data width.

- D16, D32, and D64 restrictions

  The VMEbus address and the memory address must be longword aligned (quadword aligned for D64), and the lowest 8 address bits [7:0] must match exactly.

  The requested byte count must be in multiples of the data size (multiples of 2, 4, and 8 for D16, D32, and D64, respectively).

- Further D64 restrictions

  If the VMEbus address is aligned on a 2 KB boundary, the memory address must also be aligned on a 2 KB boundary. This restriction will be removed in a future release of the operating system.

Note that you can use the `valloc` function to allocate memory aligned to a page boundary, as described in the `valloc`(3) reference page.

For the best DMA performance, the VMEbus address and the memory address should be aligned to a 256-byte boundary for D16 and D32 DMA transfers, or to a 2048-byte boundary for D64 DMA transfers.

The Alpha VME 2100 system in an SMP environment emulates DMA transfers using PIO operations instead of using an MBLT hardware DMA engine. The VMEbus adapter on this system requires three I/O accesses to be atomic to start the DMA engine. These I/O operations cannot be guaranteed to be atomic in an SMP environment. Uniprocessor systems use the MBLT hardware DMA engine.

2.2.9 Using the Realtime Interrupt-Handling Routine rt_post_callout

Interrupt service interfaces (ISIs) executing at SPLDEVRT (SPL 6) must not call kernel routines directly. The `rt_post_callout` routine allows the calling process to defer execution of a function until a time when kernel routines can be invoked. The function invoked by `rt_post_callout` runs at an elevated SPL and is subject to the same restrictions as an ISI.

The syntax for the function invoked by `rt_post_callout` is as follows:

```
int (*function)(),
long arg1,
long arg2 );
```

The parameters for the `rt_post_callout` routine are as follows:

`function`	Name of the function to be invoked
`arg1`	The first argument passed to the function
`arg2`	The second argument passed to the function

If `rt_post_callout` is called again with the same function and arguments specified, then the duplicate invocation is dismissed before the first invocation has executed.

The following example is for an interrupt service interface (ISI) that runs at SPLDEVRT:

```
rt_dev_intr(unit)
        int unit;
{
 register struct rt_softc *sc = rt_softc[unit];
```

```
 rt_post_callout(user_wakeup_interface,   /* User wakeup function */
                 (long) &sc->error_recovery_flag, /* Event to wake*/
                 (long) NULL);            /* Unused argument      */
 return;
}
```

The following example shows a user-written function to wake up an event called by the `rt_post_callout` routine:

```
void user_wakeup_interface ( arg1, arg2 )
long arg1;
long arg2;
{
   thread_wakeup( (vm_offset_t) arg1);
}
```

2.3 Configuring UNIVERSE II–Based Alpha VME SBCs

This section describes how to set up UNIVERSE II–based Alpha VME systems for use on the VMEbus, including how to modify attributes of the `vba_univ` kernel subsystem.

VMEbus UNIVERSE II setup allows you to run the operating system on UNIVERSE II–based Alpha VME systems. For information about installing the operating system on these systems, see the *Installation Guide*.

For information about setting up VIP/VIC-based Alpha VME systems for use on the VMEbus, see Section 2.2.

This section addresses the following topics relating to the use of the VMEbus on UNIVERSE II–based Alpha VME systems:

- Configuring the `vba_univ` subsystem (Section 2.3.1)
- Configuring PCI-to-VME address spaces (Section 2.3.2)
- Configuring a special A24/A16 PCI-to-VME window (Section 2.3.3)
- Configuring VME-to-PCI address spaces (Section 2.3.4)
- Mapping UNIVERSE II CSRs to the VMEbus (Section 2.3.5)
- Mapping a location monitor window to the VMEbus (Section 2.3.6)
- Configuring VMEbus interrupts (Section 2.3.7)
- Using VMEbus software byte swapping (Section 2.3.8)
- Sharing memory between big endian and little endian processors (Section 2.3.9)
- Performing VMEbus slave block transfers (Section 2.3.10)
- Performing VMEbus master block transfers with local DMA (Section 2.3.11)

- Using the realtime interrupt-handling routine `rt_post_callout` (Section 2.3.12)

2.3.1 Configuring the vba_univ Subsystem

This section describes how to configure the `vba_univ` kernel subsystem in order to prepare UNIVERSE II–based Alpha VME systems for use on the VMEbus.

You configure the UNIVERSE II adapter by examining the default (or current) attributes supplied for the `vba_univ` subsystem, determining which attributes (if any) you want to change, then modifying the `/etc/sysconfigtab` file on your machine. After modifying `/etc/sysconfigtab`, you must shut down and reboot the system.

Note

Do not directly edit `/etc/sysconfigtab`. Instead, use the `sysconfigdb` facility, as described in the `sysconfigdb`(8) reference page. It is recommended that you maintain private `sysconfigtab` file fragments for `vba_univ` attributes and use `sysconfigdb` switches to add (`-a -f`), delete (`-d`), or merge (`-m -f`) `vba_univ` attribute values. The example in Section 3.7 illustrates this approach. The `sys_attrs`(5) reference page provides additional guidelines for editing kernel subsystem attributes. You must always reboot after changing `vba_univ` subsystem attributes.

You can modify values for the following UNIVERSE II adapter attributes; each list item corresponds to a later subsection:

Adapter interrupt dispatch policy (Section 2.3.1.1)
Adapter PCI scatter/gather maximum size (Section 2.3.1.2)
Adapter DMA window maximum size (Section 2.3.1.3)
PCI coupled window timer value (Section 2.3.1.4)
PCI maximum retries (Section 2.3.1.5)
PCI posted write transfer count (Section 2.3.1.6)
PCI aligned burst size (Section 2.3.1.7)
VMEbus request level (Section 2.3.1.8)
VMEbus request mode (Section 2.3.1.9)
VMEbus release mode (Section 2.3.1.10)
VMEbus timeout period (Section 2.3.1.11)
VMEbus arbitration mode (Section 2.3.1.12)
VMEbus arbitration timeout period (Section 2.3.1.13)
System controller VMEbus resets (Section 2.3.1.14 and Section 2.3.1.15)
VMEbus on and off counters for MBLTs (Section 2.3.1.16)

You can also configure VMEbus windows in the following ways; each list item corresponds to a later subsection:

Configuring PCI-to-VME address spaces (Section 2.3.2)
Configuring a special A24/A16 PCI-to-VME window (Section 2.3.3)
Configuring VME-to-PCI address spaces (Section 2.3.4)
Mapping UNIVERSE II CSRs to the VMEbus (Section 2.3.5)
Mapping a location monitor window to the VMEbus (Section 2.3.6)

Table 2–4 lists the defaults supplied for various VMEbus parameters. The default values specified should provide proper VMEbus operation for most applications. Be careful when modifying these values; not all adapters support all fields.

Table 2–4: UNIVERSE II VMEbus Adapter Defaults

Parameter	Default	Meaning
VBA_ISR_Dispatch_Policy	1	Adapter interruupt dispatch policy is to process all interrupts for the current SPL (only)
VBA_Max_PCI_Sg_Size	0x20000000	Maximum PCI scatter/gather size is 512 MB
VBA_Max_DMA_Wndw_Size	0x4000000	Maximum DMA window size is 64 MB
PCI_Coupled_Wndw_Tmr	0x2	Coupled Window Timer set to hold VMEbus for 32 PCI clock cycles after a coupled transaction
PCI_Max_Retry	0xF	PCI maximum retries before signaling error set to 960 (value*64)
PCI_Posted_Wrt_On_Cnt	0x0	PCI posted write transfer count is 128 bytes
PCI_Aligned_Burst_Size	0x1	PCI aligned burst size is 64 bytes
VME_Br_Lev	0x3	Bus request level 3 for master cycles
VME_Fair_Req	0x1	VMEbus request mode is fair (not demand)
VME_Rel_Mode	0x1	Release mode is release on request (ROR)
VME_Bus_To	0x6	VMEbus timeout period is 512 microseconds
VME_Arb_Mode	0x0	Arbitration mode is round robin
VME_Arb_To	0x1	VMEbus arbitration timeout period is 16 microseconds

Table 2–4: UNIVERSE II VMEbus Adapter Defaults (cont.)

Parameter	Default	Meaning
VME_Syscon	0x1	System controller VMEbus reset is enabled
VME_Von_D64	0x4	VMEbus On counter for D64 MBLT: hold bus tenure for 2048 bytes
VME_Voff_D64	0x9	VMEbus Off counter for D64 MBLT: DMA interleave is 4 microseconds
VME_Von_D32	0x2	VMEbus On counter for D32 MBLT: hold bus tenure for 512 bytes
VME_Voff_D32	0x9	VMEbus Off counter for D32 MBLT: DMA interleave is 4 microseconds
For the special A24/A16 PCI-to-VME (PCI slave) window:		
VME_A24_A16_Wnd_Ena	1	Special A24/A16 PCI-to-VME window (64 MB) is enabled
VME_A24_A16_Wnd_WP_Ena	1	Write posting enabled to the A24/A16 window
VME_A24_A16_Wnd_Dwdth	0xF	A24/A16 window maximum data width is D32 (all quadrants)
PCI_SLSI_Base	0	Stores A24/A16 (64 MB) window base address (obtained from firmware)
VME_A24_Size	0xFF0000	Stores the size of each A24 address space within the A24/A16 window; obtainable via `sysconfig -q`, default is 16MB-64KB
VME_A16_Size	0x10000	Stores the size of each A16 address space within the A24/A16 window; obtainable via `sysconfig -q`, default is 64KB
For PCI-to-VME (PCI slave) windows 0 through 7:		
PCI_LSI_Base	0	Stores base address of the contiguous PCI dense space available for PCI-to-VME windows (obtained from firmware)
PCI_Mem_Avail	0	Stores number of bytes allocated by firmware for PCI-to-VME windows
PCI_Mem_Free	0	Stores number of bytes available for further PCI-to-VME window allocations
VME_Wnd0_Ena	1	**Window 0 is enabled:**
VME_Wnd0_VME_Address	0x80000000	VMEbus base address is 0x80000000

Table 2–4: UNIVERSE II VMEbus Adapter Defaults (cont.)

Parameter	Default	Meaning
VME_Wnd0_Size	0x08000000	Size is 128 MB
VME_Wnd0_AM_Space	2	A32 space
VME_Wnd0_AM_Usr_Sprvsr	1	User mode
VME_Wnd0_AM_Data_Prg	1	Data access
VME_Wnd0_Dwdth	2	Maximum data width is D32
VME_Wnd0_WP_Ena	1	Write posting is enabled
VME_Wnd0_Cycle_Sel	0	VMEbus single cycles only
VME_Wnd1_Ena	1	**Window 1 is enabled:**
VME_Wnd1_VME_Address	0x80000000	VMEbus base address is 0x80000000
VME_Wnd1_Size	0x08000000	Size is 128 MB
VME_Wnd1_AM_Space	2	A32 space
VME_Wnd1_AM_Usr_Sprvsr	1	User mode
VME_Wnd1_AM_Data_Prg	2	Program access
VME_Wnd1_Dwdth	2	Maximum data width is D32
VME_Wnd1_WP_Ena	1	Write posting is enabled
VME_Wnd1_Cycle_Sel	0	VMEbus single cycles only
VME_Wnd2_Ena	1	**Window 2 is enabled:**
VME_Wnd2_VME_Address	0x80000000	VMEbus base address is 0x80000000
VME_Wnd2_Size	0x08000000	Size is 128 MB
VME_Wnd2_AM_Space	2	A32 space
VME_Wnd2_AM_Usr_Sprvsr	2	Supervisory mode
VME_Wnd2_AM_Data_Prg	1	Data access
VME_Wnd2_Dwdth	2	Maximum data width is D32
VME_Wnd2_WP_Ena	1	Write posting is enabled
VME_Wnd2_Cycle_Sel	0	VMEbus single cycles only
VME_Wnd3_Ena	1	**Window 3 is enabled:**
VME_Wnd3_VME_Address	0x80000000	VMEbus base address is 0x80000000
VME_Wnd3_Size	0x08000000	Size is 128 MB
VME_Wnd3_AM_Space	2	A32 space
VME_Wnd3_AM_Usr_Sprvsr	2	Supervisory mode

Table 2–4: UNIVERSE II VMEbus Adapter Defaults (cont.)

Parameter	Default	Meaning
VME_Wnd3_AM_Data_Prg	2	Program access
VME_Wnd3_Dwdth	2	Maximum data width is D32
VME_Wnd3_WP_Ena	1	Write posting is enabled
VME_Wnd3_Cycle_Sel	0	VMEbus single cycles only
VME_Wnd4_Ena	1	**Window 4 is enabled:**
VME_Wnd4_VME_Address	0x00FF0000	VMEbus base address is 0xFF0000
VME_Wnd4_Size	0x00010000	Size is 64 KB
VME_Wnd4_AM_Space	1	A24 space
VME_Wnd4_AM_Usr_Sprvsr	1	User mode
VME_Wnd4_AM_Data_Prg	1	Data access
VME_Wnd4_Dwdth	2	Maximum data width is D32
VME_Wnd4_WP_Ena	1	Write posting is enabled
VME_Wnd4_Cycle_Sel	0	VMEbus single cycles only
VME_Wnd5_Ena	1	**Window 5 is enabled:**
VME_Wnd5_VME_Address	0x00FF0000	VMEbus base address is 0xFF0000
VME_Wnd5_Size	0x00010000	Size is 64 KB
VME_Wnd5_AM_Space	1	A24 space
VME_Wnd5_AM_Usr_Sprvsr	2	Supervisory mode
VME_Wnd5_AM_Data_Prg	1	Data access
VME_Wnd5_Dwdth	2	Maximum data width is D32
VME_Wnd5_WP_Ena	1	Write posting is enabled
VME_Wnd5_Cycle_Sel	0	VMEbus single cycles only
VME_Wnd6_Ena	0	**Window 6 is disabled by default:**
VME_Wnd6_VME_Address	0x0	
VME_Wnd6_Size	0x0	
VME_Wnd6_AM_Space	0	A16 space
VME_Wnd6_AM_Usr_Sprvsr	1	User mode
VME_Wnd6_AM_Data_Prg	1	Data access
VME_Wnd6_Dwdth	2	Maximum data width is D32
VME_Wnd6_WP_Ena	1	Write posting is enabled

Table 2–4: UNIVERSE II VMEbus Adapter Defaults (cont.)

Parameter	Default	Meaning
VME_Wnd6_Cycle_Sel	0	VMEbus single cycles only
VME_Wnd7_Ena	0	**Window 7 is disabled by default:**
VME_Wnd7_VME_Address	0x0	
VME_Wnd7_Size	0x0	
VME_Wnd7_AM_Space	0	A16 space
VME_Wnd7_AM_Usr_Sprvsr	1	User mode
VME_Wnd7_AM_Data_Prg	1	Data access
VME_Wnd7_Dwdth	2	Maximum data width is D32
VME_Wnd7_WP_Ena	1	Write posting is enabled
VME_Wnd7_Cycle_Sel	0	VMEbus single cycles only
For VME-to-PCI (VMEbus slave) windows 0 through 7:		
PCI_Wnd0_Ena	1	**Window 0 is enabled:**
PCI_Wnd0_VME_Address	0x00C00000	VMEbus base address is 0xC00000
PCI_Wnd0_Size	0x00400000	Size is 4 MB
PCI_Wnd0_AM_Space	1	A24 space
PCI_Wnd0_AM_Usr_Sprvsr	3	Both user and supervisory mode
PCI_Wnd0_AM_Data_Prg	3	Both data and program access
PCI_Wnd0_WP_Ena	1	Write posting is enabled
PCI_Wnd0_Pre_Rd_Ena	1	Prefetch reads are enabled
PCI_Wnd0_PCI64_Ena	1	PCI64 transactions are enabled
PCI_Wnd0_PCI_Lock_Ena	0	Lock is disabled (not modifiable)
PCI_Wnd1_Ena	1	**Window 1 is enabled:**
PCI_Wnd1_VME_Address	0x08000000	VMEbus base address is 0x8000000
PCI_Wnd1_Size	0x08000000	Size is 128 MB
PCI_Wnd1_AM_Space	2	A32 space
PCI_Wnd1_AM_Usr_Sprvsr	3	Both user and supervisory mode
PCI_Wnd1_AM_Data_Prg	3	Both data and program access
PCI_Wnd1_WP_Ena	1	Write posting is enabled
PCI_Wnd1_Pre_Rd_Ena	1	Prefetch reads are enabled
PCI_Wnd1_PCI64_Ena	1	PCI64 transactions are enabled

Table 2–4: UNIVERSE II VMEbus Adapter Defaults (cont.)

Parameter	Default	Meaning
PCI_Wnd1_PCI_Lock_Ena	0	Lock is disabled (not modifiable)
PCI_Wnd2_Ena	0	**Window 2 is disabled by default:**
PCI_Wnd2_VME_Address	0x0	
PCI_Wnd2_Size	0x0	
PCI_Wnd2_AM_Space	1	A24 space
PCI_Wnd2_AM_Usr_Sprvsr	3	Both user and supervisory mode
PCI_Wnd2_AM_Data_Prg	3	Both data and program access
PCI_Wnd2_WP_Ena	1	Write posting is enabled
PCI_Wnd2_Pre_Rd_Ena	1	Prefetch reads are enabled
PCI_Wnd2_PCI64_Ena	1	PCI64 transactions are enabled
PCI_Wnd2_PCI_Lock_Ena	0	Lock is disabled (not modifiable)
PCI_Wnd3_Ena	0	**Window 3 is disabled by default:**
PCI_Wnd3_VME_Address	0x0	
PCI_Wnd3_Size	0x0	
PCI_Wnd3_AM_Space	1	A24 space
PCI_Wnd3_AM_Usr_Sprvsr	3	Both user and supervisory mode
PCI_Wnd3_AM_Data_Prg	3	Both data and program access
PCI_Wnd3_WP_Ena	1	Write posting is enabled
PCI_Wnd3_Pre_Rd_Ena	1	Prefetch reads are enabled
PCI_Wnd3_PCI64_Ena	1	PCI64 transactions are enabled
PCI_Wnd3_PCI_Lock_Ena	0	Lock is disabled (not modifiable)
PCI_Wnd4_Ena	0	**Window 4 is disabled by default:**
PCI_Wnd4_VME_Address	0x0	
PCI_Wnd4_Size	0x0	
PCI_Wnd4_AM_Space	1	A24 space
PCI_Wnd4_AM_Usr_Sprvsr	3	Both user and supervisory mode
PCI_Wnd4_AM_Data_Prg	3	Both data and program access
PCI_Wnd4_WP_Ena	1	Write posting is enabled
PCI_Wnd4_Pre_Rd_Ena	1	Prefetch reads are enabled
PCI_Wnd4_PCI64_Ena	1	PCI64 transactions are enabled

Table 2–4: UNIVERSE II VMEbus Adapter Defaults (cont.)

Parameter	Default	Meaning
PCI_Wnd4_PCI_Lock_Ena	0	Lock is disabled (not modifiable)
PCI_Wnd5_Ena	0	**Window 5 is disabled by default:**
PCI_Wnd5_VME_Address	0x0	
PCI_Wnd5_Size	0x0	
PCI_Wnd5_AM_Space	1	A24 space
PCI_Wnd5_AM_Usr_Sprvsr	3	Both user and supervisory mode
PCI_Wnd5_AM_Data_Prg	3	Both data and program access
PCI_Wnd5_WP_Ena	1	Write posting is enabled
PCI_Wnd5_Pre_Rd_Ena	1	Prefetch reads are enabled
PCI_Wnd5_PCI64_Ena	1	PCI64 transactions are enabled
PCI_Wnd5_PCI_Lock_Ena	0	Lock is disabled (not modifiable)
PCI_Wnd6_Ena	0	**Window 6 is disabled by default:**
PCI_Wnd6_VME_Address	0x0	
PCI_Wnd6_Size	0x0	
PCI_Wnd6_AM_Space	1	A24 space
PCI_Wnd6_AM_Usr_Sprvsr	3	Both user and supervisory mode
PCI_Wnd6_AM_Data_Prg	3	Both data and program access
PCI_Wnd6_WP_Ena	1	Write posting is enabled
PCI_Wnd6_Pre_Rd_Ena	1	Prefetch reads are enabled
PCI_Wnd6_PCI64_Ena	1	PCI64 transactions are enabled
PCI_Wnd6_PCI_Lock_Ena	0	Lock is disabled (not modifiable)
PCI_Wnd7_Ena	0	**Window 7 is disabled by default:**
PCI_Wnd7_VME_Address	0x0	
PCI_Wnd7_Size	0x0	
PCI_Wnd7_AM_Space	1	A24 space
PCI_Wnd7_AM_Usr_Sprvsr	3	Both user and supervisory mode
PCI_Wnd7_AM_Data_Prg	3	Both data and program access
PCI_Wnd7_WP_Ena	1	Write posting is enabled
PCI_Wnd7_Pre_Rd_Ena	1	Prefetch reads are enabled
PCI_Wnd7_PCI64_Ena	1	PCI64 transactions are enabled

Table 2–4: UNIVERSE II VMEbus Adapter Defaults (cont.)

Parameter	Default	Meaning
PCI_Wnd7_PCI_Lock_Ena	0	Lock is disabled (not modifiable)
For UNIVERSE II CSR and location monitor window mapping:		
CSR_Ena	1	**UNIVERSE II CSR mapping is enabled:**
CSR_VME_Address	0xFFFF0000	VMEbus base address is 0xFFFF0000
CSR_AM_Space	2	A32 space
CSR_AM_Usr_Sprvsr	2	Supervisory mode
CSR_AM_Data_Prg	3	Both program and data access
LM_Ena	0	**Location monitor mapping is disabled by default:**
LM_VME_Address	0xFFFF1000	VMEbus base address is 0xFFFF1000
LM_AM_Space	2	A32 space
LM_AM_Usr_Sprvsr	2	Supervisory mode
LM_AM_Data_Prg	3	Both program and data access

Table 2–5 lists VMEbus interrupt parameters and their initial defaults. These defaults are later overwritten by system priority level (SPL) values supplied by the platform. See the SPL values listed in Table 2–6, or query the values at run time using the command `sysconfig -q vba_univ`.

Table 2–5: UNIVERSE II VMEbus Interrupt Initial Defaults

Parameter	Default	Meaning
Irq0_SPL	4	VMEbus IRQ level to system SPL map
Irq1_SPL	4	VMEbus IRQ 1 to SPL SPLDEVHIGH
Irq2_SPL	4	VMEbus IRQ 2 to SPL SPLDEVHIGH
Irq3_SPL	4	VMEbus IRQ 3 to SPL SPLDEVHIGH
Irq4_SPL	4	VMEbus IRQ 4 to SPL SPLDEVHIGH
Irq5_SPL	4	VMEbus IRQ 5 to SPL SPLDEVHIGH
Irq6_SPL	4	VMEbus IRQ 6 to SPL SPLDEVHIGH
Irq7_SPL	4	VMEbus IRQ 7 to SPL SPLDEVHIGH
Adapt_Blk_SPL	4	Adapter resource blocking SPL SPLDEVHIGH

2.3.1.1 Specifying the Adapter Interrupt Dispatch Policy

You can specify one of the following values for the adapter interrupt dispatch policy (parameter VBA_ISR_Dispatch_Policy):

1	Process all interrupts for the current SPL (default)
2	Process all interrupts for the current SPL, then check for and process additional interrupts once

2.3.1.2 Specifying the Adapter PCI Scatter/Gather Maximum Size

You can specify a multiple of 64 KB (0x10000) up to 512 MB (0x20000000) for the adapter PCI scatter/gather maximum size (parameter VBA_Max_PCI_Sg_Size). The default is 512 MB.

If the combined amount of scatter/gather resources needed to map all enabled VME-to-PCI windows exceeds the value of VBA_Max_PCI_Sg_Size, the adapter will not be configured. You can use the VBA_Max_PCI_Sg_Size parameter to constrain the consumption of PCI scatter/gather resources.

2.3.1.3 Specifying the Adapter DMA Window Maximum Size

You can specify one of the following values for the adapter DMA window maximum size (parameter VBA_Max_DMA_Wndw_Size). This value determines the amount of scatter/gather resources allocated for the DMA engine during adapter initialization. If the amount of scatter/gather resources needed for a requested DMA transfer exceeds the value of VBA_Max_DMA_Wndw_Size, the DMA transfer will be broken up into segments and the scatter/gathers will be reloaded for each segment.

You can use the VBA_Max_DMA_Wndw_Size parameter to constrain the consumption of DMA scatter/gather resources or to throttle DMA transfers (reducing granularity to force reloads).

This software resource constraint is independent of the adapter's hardware constraint on transfer size (up to 16 MB minus 2 KB of data without processor intervention).

0x2000	8 KB
0x4000	16 KB
0x8000	32 KB
0x10000	64 KB
0x20000	128 KB
0x40000	256 KB

0x80000	512 KB
0x100000	1 MB
0x200000	2 MB
0x400000	4 MB
0x800000	8 MB
0x1000000	16 MB
0x2000000	32 MB
0x4000000	64 MB (default)
0x8000000	128 MB
0x10000000	256 MB

2.3.1.4 Specifying the PCI Coupled Window Timer Value

You can specify one of the following values for the PCI coupled window timer value (parameter `PCI_Coupled_Wndw_Tmr`). This value is stored in the PCI Miscellaneous Register (LMISC).

The Universe II adapter uses the coupled window timer to determine how long to hold ownership of the VMEbus on behalf of the PCI Slave Channel after processing a coupled transaction. The timer is restarted each time the Universe II processes a coupled transaction. If this timer expires, then the PCI Slave Channel releases the VME Master Interface.

0x0	Disable Coupled Window Timer (CWT)
0x1	CWT = 16 PCI clock cycles
0x2	CWT = 32 PCI clock cycles (default)
0x3	CWT = 64 PCI clock cycles
0x4	CWT = 128 PCI clock cycles
0x5	CWT = 256 PCI clock cycles
0x6	CWT = 512 PCI clock cycles

2.3.1.5 Specifying the PCI Maximum Retries

You can specify one of the following values for the number of PCI maximum retries before signaling errors (parameter `PCI_Max_Retry`). This value is stored in the Master Control Register (MAST_CTL).

0x0	Retry forever (on PCI)
0x1	Retry 64 times

0x2	Retry 128 times
0x3	Retry 192 times
0x4	Retry 256 times
0x5	Retry 320 times
0x6	Retry 384 times
0x7	Retry 448 times
0x8	Retry 512 times
0x9	Retry 576 times
0xA	Retry 640 times
0xB	Retry 704 times
0xC	Retry 768 times
0xD	Retry 832 times
0xE	Retry 896 times
0xF	Retry 960 times (default)

2.3.1.6 Specifying the PCI Posted Write Transfer Count

You can specify one of the following values for the PCI posted write transfer count (parameter `PCI_Posted_Wrt_On_Cnt`). This value is stored in the Master Control Register (MAST_CTL).

0x0	Posted write transfer count = 128 bytes (default)
0x1	Posted write transfer count = 256 bytes
0x2	Posted write transfer count = 512 bytes
0x3	Posted write transfer count = 1024 bytes
0x4	Posted write transfer count = 2048 bytes
0x5	Posted write transfer count = 4096 bytes

2.3.1.7 Specifying the PCI Aligned Burst Size

You can specify one of the following values for the PCI aligned burst size (parameter `PCI_Aligned_Burst_Size`). This value is stored in the Master Control Register (MAST_CTL).

0x0	PCI aligned burst size = 32 bytes
0x1	PCI aligned burst size = 64 bytes (default)
0x2	PCI aligned burst size = 128 bytes

2.3.1.8 Specifying the VMEbus Request Level

You can specify one of the following values for the VMEbus request level (parameter `VME_Br_Lev`). This value is stored in the Master Control Register (MAST_CTL).

0x0	VMEbus request level BR0
0x1	VMEbus request level BR1
0x2	VMEbus request level BR2
0x3	VMEbus request level BR3 (default)

2.3.1.9 Specifying the VMEbus Request Mode

You can specify one of the following values for the VMEbus request mode (parameter `VME_Fair_Req`). This value is stored in the Master Control Register (MAST_CTL).

0x0	Request mode is demand
0x1	Request mode is fair (default)

2.3.1.10 Specifying the VMEbus Release Mode

You can specify one of the following values for the release mode (parameter `VME_Rel_Mode`). This value is stored in the Master Control Register (MAST_CTL).

0x0	Release when done, RWD
0x1	Release on request, ROR (default)

2.3.1.11 Specifying the VMEbus Timeout Period

You can specify one of the following values for the VMEbus timeout period (parameter `VME_Bus_To`). This value is stored in the Miscellaneous Control Register (MISC_CTL).

0x0	Timeouts are disabled
0x1	Timeout = 16 microseconds
0x2	Timeout = 32 microseconds
0x3	Timeout = 64 microseconds
0x4	Timeout = 128 microseconds

0x5	Timeout = 256 microseconds
0x6	Timeout = 512 microseconds (default)

2.3.1.12 Specifying the VMEbus Arbitration Mode

You can specify one of the following values for the VMEbus arbitration mode (parameter `VME_Arb_Mode`). This value is stored in the Miscellaneous Control Register (MISC_CTL). This parameter is applicable only when the VMEbus adapter is configured to be the system controller.

0x0	UNIVERSE II performs round-robin VMEbus arbitration (default)
0x1	UNIVERSE II performs priority VMEbus arbitration

2.3.1.13 Specifying the VMEbus Arbitration Timeout Period

You can specify one of the following values for the VMEbus arbitration timeout period (parameter `VME_Arb_To`). This value is stored in the Miscellaneous Control Register (MISC_CTL).

0x0	Timeouts are disabled
0x1	Timeout = 16 microseconds (default)
0x2	Timeout = 256 microseconds

2.3.1.14 Specifying System Controller VMEbus Resets

You can specify one of the following values to indicate whether or not the adapter should issue VMEbus resets if it is the system controller (parameter `VME_Syscon`). This value is stored in the Miscellaneous Control Register (MISC_CTL).

For Alpha VME SBCs, in addition to specifying a value from this list, you must set the configuration switches to indicate whether or not the SBC is the VMEbus system controller. See the SBC's installation guide for information on setting the module configuration switches.

The VMEbus backplane must have only one system controller. The system controller must be electrically the first module in the VMEbus backplane and in most systems must be in the first VMEbus slot.

0x0	Do not issue VMEbus resets if system controller
0x1	Issue VMEbus resets if system controller (default)

The values specified interact with the VMEbus initialization code to determine whether a VMEbus reset is issued when the VMEbus adapter is being configured. If the value is set to 1 and the system being booted is the system controller, as determined by the VMEbus initialization code, a VMEbus reset is issued. If you do not want a VMEbus reset issued during VMEbus adapter configuration, set the value to 0 (zero). These values pertain only to the system controller.

If the system controller is configured to issue a VMEbus reset during adapter initialization, and other processor modules are installed in the VMEbus backplane, boot the system controller first to allow devices and processor modules to perform their bus reset actions.

2.3.1.15 Special Considerations for VMEbus Resets

The system controller should always be the initiator of VMEbus resets. However, under certain error conditions, other VMEbus adapter modules may invoke a VMEbus reset. Modules installed in the VMEbus backplane react to bus resets differently. Some modules, if configured, perform a module reset. Some may have their VMEbus interface reset to a power-up state without notification to the operating system. This could leave the VMEbus adapters in an unconfigured state, cause unwanted effects to the operating system and its device drivers, and cause VMEbus errors to occur. Other VMEbus adapters on the VMEbus may accept VMEbus resets and attempt to reconfigure themselves to the hardware context they were running before the bus reset occurred. However, device drivers expecting interrupts may not receive them and I/O hardware operations may be canceled by the VMEbus reset without notification to the device driver. There is also a potential for data corruption to occur when the VMEbus adapter is reset during an I/O operation.

It is recommended that the system controller be the initiator of VMEbus resets during adapter initialization. If the system controller is not controlled by a processor, then a power-up sequence should cause all VMEbus adapters and devices to be reset. All modules on the VMEbus should perform a module reset upon detection of a bus reset. VMEbus adapters that are not the system controller and that are running an operating system should be shut down in an orderly fashion prior to the system controller being booted. These VMEbus adapters should be rebooted after the system controller has been booted, providing that the system controller is to be used and controlled by a processor.

For Alpha VME SBCs, it is recommended that nodes that are not the system controller have their module configuration switch 3 set to Closed (resets the SBC module on VMEbus reset signal). When the VMEbus is reset, and the module switch is set to accept a VMEbus reset, nonsystem controller modules take a boot action and are reset to a powered state.

If the SBC module configuration switch 3 is set to Open (does not reset the SBC module on VMEbus reset signal), the VMEbus adapter software will receive a VMEbus reset interrupt upon detection of a bus reset. The VMEbus reset signal initializes the VMEbus adapter to its power-up state. The VMEbus reset interrupt service interface displays the following message on the console terminal:

```
vba0 reset_inter:  VMEbus reset detected
```

The interrupt service interface then initializes the VMEbus adapter to its defaults and enables any previously enabled interrupt enable bits.

Do not set the SBC module configuration switch 3 to Open without considering the following side effects of receiving a VMEbus reset: device drivers expecting interrupts may not receive them and I/O hardware operations may be canceled by the VMEbus reset without notification to the device drivers. There is potential risk of data corruption depending upon I/O activity at the time a bus reset occurred.

2.3.1.16 Specifying the VMEbus On and Off Counters for MBLTs

You can specify one of the following values for the VMEbus On Counter for D64 MBLTs (parameter `VME_Von_D64`) or the VMEbus On Counter for D32 MBLTs (parameter `VME_Von_D32`). This value is stored in the DMA General Control and Status Register (DGCS).

0x0	All bytes transferred until done
0x1	256-byte boundary
0x2	512-byte boundary (D32 MBLT default)
0x3	1024-byte boundary
0x4	2048-byte boundary (D64 MBLT default)
0x5	4096-byte boundary
0x6	8192-byte boundary
0x7	16384-byte boundary

You can specify one of the following values for the VMEbus Off Counter for D64 MBLTs (parameter `VME_Voff_D64`) or the VMEbus Off Counter for D32 MBLTs (parameter `VME_Voff_D32`). This value is stored in the DMA General Control and Status Register (DGCS).

0x0	0 microseconds between VMEbus tenures
0x1	16 microseconds between VMEbus tenures
0x2	32 microseconds between VMEbus tenures

0x3	64 microseconds between VMEbus tenures
0x4	128 microseconds between VMEbus tenures
0x5	256 microseconds between VMEbus tenures
0x6	512 microseconds between VMEbus tenures
0x7	1024 microseconds between VMEbus tenures
0x8	2 microseconds between VMEbus tenures
0x9	4 microseconds between VMEbus tenures (default)
0xA	8 microseconds between VMEbus tenures

2.3.2 Configuring PCI-to-VME Address Spaces

As part of configuring the `vba_univ` kernel subsystem, you can configure up to eight PCI-to-VME (PCI slave) windows, numbered 0 through 7, for your system. Additionally, you can map a special 64 MB window for VMEbus A24 and A16 accesses.

By default, the following PCI-to-VME windows are provided on your system:

Window 0 - enabled	VMEbus base address 0x80000000, 128 MB, A32 user data
Window 1 - enabled	VMEbus base address 0x80000000, 128 MB, A32 user program
Window 2 - enabled	VMEbus base address 0x80000000, 128 MB, A32 supervisory data
Window 3 - enabled	VMEbus base address 0x80000000, 128 MB, A32 supervisory program
Window 4 - enabled	VMEbus base address 0x00FF0000, 64 KB, A24 user data
Window 5 - enabled	VMEbus base address 0x00FF0000, 64 KB, A24 supervisory data
Window 6 - disabled	VMEbus base address 0x00000000, 0, A16 user data
Window 7 - disabled	VMEbus base address 0x00000000, 0, A16 user data
A24/A16 window - enabled	64 MB (four equal quadrants for user data, user program, supervisory data, supervisory program), top 64 KB per quadrant window is A16 (only quadrants 0 and 2 used for A16)

Firmware allocates between 512 MB (minimum) and 960 MB (maximum) of contiguous PCI dense space for PCI-to-VME windows 0 through 7, based on what is configured in the system, and an additional, separate 64 MB for the special A24/A16 window.

The default windows 0 through 3 consume 512 MB; the default windows 4 and 5 consume 128 KB. Windows 6 and 7 can be used to map to other VMEbus address spaces, module switches, semaphores, location monitors, and so on. (However, if your configuration requires more PCI resources than are available, the adapter will not be configured.)

Between the special 64 MB A24/A16 window and the eight other windows, all of A16 and A24 space is available for access. The CPU can access a 128 MB window of A32 space with the default configuration. You have the ability to increase or decrease the size of the windows, change the VMEbus addresses and modifiers, and specify additional VMEbus windows.

Note

When configuring PCI-to-VME address spaces, you must ensure that all VMEbus devices to which the CPU will perform I/O are configured within one or more of the PCI-to-VME windows. If window sizes, VMEbus addresses, or VMEbus address modifiers are changed at a later point, you must ensure that the VMEbus devices remain within the PCI-to-VME windows.

During system initialization, if the special A24/A16 PCI-to-VME window is enabled (`vba_univ` parameter `VME_A24_A16_Wnd_Ena` equals 1), the UNIVERSE II adapter support code obtains (from firmware) the PCI address of the 64 MB window that will be used for VMEbus A24 and A16 accesses and configures the window to match your `vba_univ` attribute settings. For more information about configuring the A24/A16 PCI-to-VME window, see Section 2.3.3.

The UNIVERSE II adapter support code then obtains (from firmware) the PCI start and end addresses of the contiguous PCI dense space available for mapping PCI-to-VME windows 0 through 7. If enough PCI dense space is available, windows 0 through 7 are then configured to match your `vba_univ` attribute settings.

For hardware reasons, PCI-to-VME windows 0 and 4 must be configured on a 4 KB boundary, and their sizes must be a multiple of 4 KB. The remaining six windows must be configured on a 64 KB boundary, and their sizes must be a multiple of 64 KB. The sizes of all windows together must not exceed the limit provided in firmware.

Each PCI-to-VME window has the following configurable parameters, which you can modify in the form of `vba_univ` subsystem attributes:

Window enabled or disabled (Section 2.3.2.1)
VMEbus base address (Section 2.3.2.2)
Window size (Section 2.3.2.3)

VMEbus address modifiers (Section 2.3.2.4)
VMEbus maximum data width (Section 2.3.2.5)
Write posting enabled or disabled (Section 2.3.2.6)
VMEbus cycle type (Section 2.3.2.7)

When mapping to the VMEbus to fulfill a request, UNIVERSE II support code searches PCI-to-VME windows 0 through 7 in numerically ascending order, comparing the VMEbus address attributes in the request to the configured attributes of each window. The first window that satisfies the request is used. If none of the windows 0 through 7 satisfies the request, the support code checks against the special A24/A16 PCI-to-VME window.

Note that for A24 and A16 access, the support code's VMEbus mapping algorithm allows windows 0 through 7 to take precedence over the special A24/A16 window. If you want to guarantee that CSR accesses are mapped through the special A24/A16 window, you must manipulate your system's PCI-to-VME window attributes such that the CSR mappings fall through to the special window.

2.3.2.1 Enabling or Disabling a PCI-to-VME Window

To enable or disable a PCI-to-VME window, you can specify one of the following values to the `VME_Wndn_Ena` attribute for that window. This value is stored in the PCI Slave Image Control Register corresponding to the PCI-to-VME window number (LSIn_CTL).

0x0	Window is disabled (default for windows 6 and 7)
0x1	Window is enabled (default for windows 0 through 5)

2.3.2.2 Specifying a PCI-to-VME Window VMEbus Base Address

To establish the VMEbus base address for a PCI-to-VME window, you specify a hexadecimal address value to the `VME_Wndn_VME_Address` attribute for that window. The value can be in the range 0x0 to 0xFFFFFFFF, but the window base address and its associated size (`VME_Wndn_Size`) must fall within the addressable range of the VMEbus address space (A32, A24, or A16) selected for that window.

Windows 0 and 4 must be configured on 4 KB boundaries; the remaining six windows must be configured on 64 KB boundaries.

2.3.2.3 Specifying a PCI-to-VME Window Size

To establish the size for a PCI-to-VME window, you specify a hexadecimal size value to the `VME_Wndn_Size` attribute for that window. The value can be in the range 0x0 to 0xFFFFFFFF, but the window base address (`VME_Wndn_VME_Address`) and its associated size must fall within the

addressable range of the VMEbus address space (A32, A24, or A16) selected for that window.

Windows 0 and 4 must be sized to a multiple of 4 KB; the remaining six windows must be sized to a multiple of 64 KB.

2.3.2.4 Specifying PCI-to-VME Window VMEbus Address Modifiers

To select the VMEbus address space for a PCI-to-VME window, you can specify one of the following values to the `VME_Wndn_AM_Space` attribute for that window, 0 through 7, to select the VMEbus address space for that window. This value is stored in the PCI Slave Image Control Register corresponding to the PCI-to-VME window number (LSIn_CTL).

0x0	A16 address space
0x1	A24 address space
0x2	A32 address space

To select user or supervisory mode for a PCI-to-VME window, you can specify one of the following values to the `VME_Wndn_AM_Usr_Sprvsr` attribute for that window. This value is stored in the PCI Slave Image Control Register corresponding to the PCI-to-VME window number (LSIn_CTL).

0x1	User mode
0x2	Supervisory mode

To select data or program access for a PCI-to-VME window, you can specify one of the following values to the `VME_Wndn_AM_Data_Prg` attribute for that window. This value is stored in the PCI Slave Image Control Register corresponding to the PCI-to-VME window number (LSIn_CTL).

0x1	Data access
0x2	Program access

2.3.2.5 Specifying a PCI-to-VME Window VMEbus Maximum Data Width

To select the VMEbus maximum data width for a PCI-to-VME window, you can specify one of the following values to the `VME_Wndn_AM_Dwdth` attribute for that window. This value is stored in the PCI Slave Image Control Register corresponding to the PCI-to-VME window number (LSIn_CTL).

0x0	VMEbus maximum data width = 8 bits
0x1	VMEbus maximum data width = 16 bits

0x2	VMEbus maximum data width = 32 bits (default)
0x3	VMEbus maximum data width = 64 bits

2.3.2.6 Specifying PCI-to-VME Window Write Posting

To enable or disable write posting for a PCI-to-VME window, you can specify one of the following values to the `VME_Wndn_WP_Ena` attribute for that window. This value is stored in the PCI Slave Image Control Register corresponding to the PCI-to-VME window number (LSIn_CTL).

0x0	Write posting is disabled
0x1	Write posting is enabled (default)

2.3.2.7 Specifying a PCI-to-VME Window VMEbus Cycle Type

To select the VMEbus cycle type for a PCI-to-VME window, you can specify one of the following values to the `VME_Wndn_Cycle_Sel` attribute for that window. This value is stored in the PCI Slave Image Control Register corresponding to the PCI-to-VME window number (LSIn_CTL).

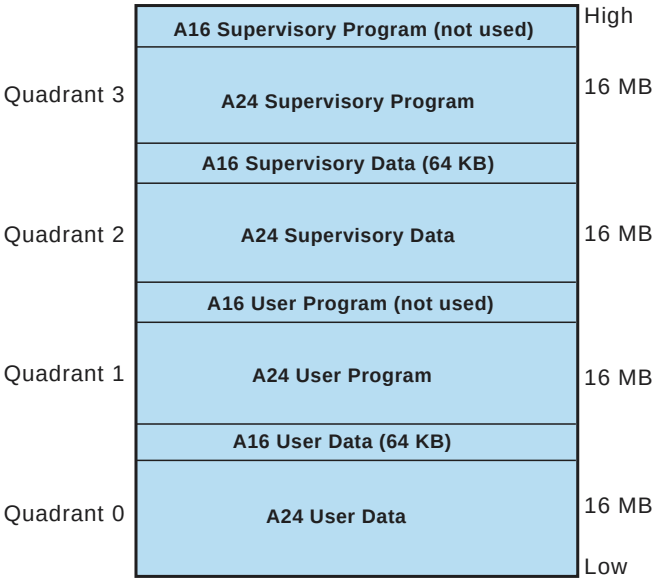
0x0	Single cycles only (default)
0x1	Single cycles and block transfers

2.3.3 Configuring a Special A24/A16 PCI-to-VME Window

As part of configuring the `vba_univ` kernel subsystem, you can configure up to eight PCI-to-VME (PCI slave) windows for your system. Additionally, you can map a special A24/A16 PCI-to-VME window, 64 MB in size, for VMEbus A24 and A16 accesses.

The 64 MB window (64 MB aligned) is subdivided into four 16 MB windows. The top 64 KB of each 16 MB quadrant is allocated for VMEbus A16 accesses. The remaining 16 MB minus 64 KB of each quadrant is allocated for VME A24 accesses.

By default, the four quadrants of the 64 MB window are set up with the following VMEbus address-modifier attributes. Note that only quadrants 0 and 2 are used for A16 access.



For example, an A16 supervisory data access would map to the top 64 KB of quadrant 2. An A24 user data access would map to the bottom 16 MB minus 64 KB of quadrant 0.

The special A24/A16 PCI-to-VME window has the following configurable parameters, which you can modify in the form of `vba_univ` subsystem attributes:

Window enabled or disabled (Section 2.3.3.1)
Write posting enbled or disabled (Section 2.3.3.2)
VMEbus maximum data width (Section 2.3.3.3)

During system initialization, if the special A24/A16 PCI-to-VME window is enabled (`vba_univ` parameter `VME_A24_A16_Wnd_Ena` equals 1), the UNIVERSE II adapter interface obtains (from firmware) the PCI address of the 64 MB window that will be used for VMEbus A24 and A16 accesses and configures the window to match your `vba_univ` attribute settings.

2.3.3.1 Enabling or Disabling the A24/A16 Window

You can specify one of the following values to the `VME_A24_A16_Wnd_Ena` attribute to enable or disable the special A24/A16 PCI-to-VME window. This value is stored in the Special PCI Slave Image Register (SLSI).

0x0	A24/A16 window is disabled
0x1	A24/A16 window is enabled (default)

2.3.3.2 Specifying A24/A16 Window Write Posting

You can specify one of the following values to the VME_A24_A16_Wnd_WP_Ena attribute to enable or disable write posting to the A24/A16 window. This value is stored in the Special PCI Slave Image Register (SLSI).

0x0	Write posting is disabled
0x1	Write posting is enabled (default)

2.3.3.3 Specifying the A24/A16 Window VMEbus Maximum Data Width

You can specify a 4-bit value from 0x0 to 0xF to the VME_A24_A16_Wnd_Dwdth attribute to select the A24/A16 window VMEbus maximum data width for each quadrant. This value is stored in the Special PCI Slave Image Register (SLSI).

Each bit selects D16 (0) or D32 (1) width for the corresponding quadrant, as follows:

Q3	Q2	Q1	Q0	
0	0	0	0	16-bit data width (D16)
1	1	1	1	32-bit data width (D32)

ZK-1565U-AI

For example, the value 0x0 (bit value 0000) selects D16 for all quadrants and the value 0xA (1010) selects D16 for quadrants 0 and 2 and D32 for quadrants 1 and 3. The default, 0xF (1111), selects D32 for all quadrants.

2.3.4 Configuring VME-to-PCI Address Spaces

As part of configuring the vba_univ kernel subsystem, you can configure up to eight VME-to-PCI (VMEbus slave) windows, numbered 0 through 7, to be used for VMEbus slave accesses in your system.

By default, the following VME-to-PCI windows are provided on your system:

Window 0 - enabled	VMEbus base address 0x00C00000, 4 MB, A24 user/supervisory data/program; write posting, prefetching, and PCI64 enabled
Window 1 - enabled	VMEbus base address 0x08000000, 128 MB, A32 user/supervisory data/program; write posting, prefetching, and PCI64 enabled
Window 2 - disabled	VMEbus base address 0x00000000, 0, A24 user/supervisory data/program; write posting, prefetching, and PCI64 enabled
Window 3 - disabled	VMEbus base address 0x00000000, 0, A24 user/supervisory data/program; write posting, prefetching, and PCI64 enabled
Window 4 - disabled	VMEbus base address 0x00000000, 0, A24 user/supervisory data/program; write posting, prefetching, and PCI64 enabled
Window 5 - disabled	VMEbus base address 0x00000000, 0, A24 user/supervisory data/program; write posting, prefetching, and PCI64 enabled
Window 6 - disabled	VMEbus base address 0x00000000, 0, A24 user/supervisory data/program; write posting, prefetching, and PCI64 enabled
Window 7 - disabled	VMEbus base address 0x00000000, 0, A24 user/supervisory data/program; write posting, prefetching, and PCI64 enabled

Other windows can be enabled, or enabled windows can be reconfigured. All windows must be at least 64 MB in size. Windows 0 and 4 must be configured on an 8 KB boundary and must be sized to a multiple of 8 KB (minimum 64 KB), in order to line up with the PCI scatter/gather mapping register on Alpha based platforms. The remaining six windows must be configured on a 64 KB boundary and must be sized to a multiple of 64 KB. The sizes of all windows together must not exceed the total amount of resources available in the system for VME-to-PCI mapping. The number of VME-to-PCI windows enabled in the system, their sizes, and the amount of memory in the system determines the PCI resources needed. The maximum memory provided for VME-to-PCI mapping resources is determined by the `VBA_Max_PCI_Sg_Size` adapter attribute; the default is 512 MB.

Each VME-to-PCI window has the following configurable parameters, which you can modify in the form of `vba_univ` subsystem attributes:

Window enabled or disabled (Section 2.3.4.1)
VMEbus base address (Section 2.3.4.2)
Window size (Section 2.3.4.3)
VMEbus address modifiers (Section 2.3.4.4)
Write posting enabled or disabled (Section 2.3.4.5)
Prefetch reads enabled or disabled (Section 2.3.4.6)
64-bit PCI bus transactions enabled or disabled (Section 2.3.4.7)

2.3.4.1 Enabling or Disabling a VME-to-PCI Window

To enable or disable a VME-to-PCI window, you can specify one of the following values to the PCI_Wndn_Ena attribute for that window. This value is stored in the VMEbus Slave Image Control Register corresponding to the VME-to-PCI window number (VSIn_CTL).

0x0	Window is disabled (default for windows 2 through 7)
0x1	Window is enabled (default for windows 0 and 1)

2.3.4.2 Specifying a VME-to-PCI Window VMEbus Base Address

To establish the VMEbus base address for a VME-to-PCI window, you specify a hexadecimal address value to the PCI_Wndn_VME_Address attribute for that window. The value can be in the range 0x0 to 0xFFFFFFFF, but the window base address and its associated size (PCI_Wndn_Size) must fall within the addressable range of the VMEbus address space (A32 or A24) selected for that window.

Windows 0 and 4 must be configured on 8 KB boundaries to line up with the PCI scatter/gather mapping register on Alpha based systems; the remaining six windows must be configured on 64 KB boundaries.

2.3.4.3 Specifying a VME-to-PCI Window Size

To establish the size for a VME-to-PCI window, you specify a hexadecimal size value to the PCI_Wndn_Size attribute for that window. The value can be in the range 0x0 to 0xFFFFFFFF, but the window base address (PCI_Wndn_VME_Address) and its associated size must fall within the addressable range of the VMEbus address space (A32 or A24) selected for that window.

All windows must be at least 64 KB in size. Windows 0 and 4 must be sized to a multiple of 8 KB; the remaining six windows must be sized to a multiple of 64 KB.

2.3.4.4 Specifying VME-to-PCI Window VMEbus Address Modifiers

To select the VMEbus address space for a VME-to-PCI window, you can specify one of the following values to the PCI_Wndn_AM_Space attribute for that window. This value is stored in the VMEbus Slave Image Control Register corresponding to the VME-to-PCI window number (VSIn_CTL).

0x1	A24 address space
0x2	A32 address space

You can specify one of the following values to the PCI_Wndn_AM_Usr_Sprvsr attribute for a VME-to-PCI window (0 through 7) to select user mode, supervisory mode, or both for that window. This value is stored in the VMEbus Slave Image Control Register corresponding to the VME-to-PCI window number (VSIn_CTL).

0x1	User mode
0x2	Supervisory mode
0x3	Both user and supervisory mode (default)

You can specify one of the following values to the PCI_Wndn_AM_Data_Prg attribute for a VME-to-PCI window (0 through 7) to select data access, program access, or both for that window. This value is stored in the VMEbus Slave Image Control Register corresponding to the VME-to-PCI window number (VSIn_CTL).

0x1	Data access
0x2	Program access
0x3	Both data and program access (default)

2.3.4.5 Specifying VME-to-PCI Window Write Posting

To enable or disable write posting for a VME-to-PCI window, you can specify one of the following values to the PCI_Wndn_WP_Ena attribute for that window. This value is stored in the VMEbus Slave Image Control Register corresponding to the VME-to-PCI window number (VSIn_CTL).

0x0	Write posting is disabled
0x1	Write posting is enabled (default)

2.3.4.6 Specifying VME-to-PCI Window Prefetch Reads

To enable or disable prefetch reads for a VME-to-PCI window, you can specify one of the following values to the PCI_Wndn_Pre_Rd_Ena attribute for that window. This value is stored in the VMEbus Slave Image Control Register corresponding to the VME-to-PCI window number (VSIn_CTL).

0x0	Prefetch reads are disabled
0x1	Prefetch reads are enabled (default)

2.3.4.7 Specifying VME-to-PCI Window 64-Bit PCI Bus Transactions

To enable or disable 64-bit PCI bus transactions for a VME-to-PCI window, you can specify one of the following values to the `PCI_Wndn_PCI64_Ena` attribute for that window. This value is stored in the VMEbus Slave Image Control Register corresponding to the VME-to-PCI window number (VSIn_CTL).

0x0	64-bit PCI bus transactions are disabled
0x1	64-bit PCI bus transactions are enabled (default)

Note

In order for 64-bit PCI bus transactions to be enabled, the PCI Bus Size (LCLSIZE) bit must be set in the Miscellaneous Status Register (MISC_STAT). If LCLSIZE is clear, the value of the `PCI_Wndn_PCI64_Ena` attribute is ignored.

2.3.5 Mapping UNIVERSE II CSRs to the VMEbus

As part of configuring the `vba_univ` kernel subsystem, you can map UNIVERSE II CSRs (control and status registers) to the VMEbus for your system. UNIVERSE II CSRs occupy a 4 KB window and can be enabled to support four module switches and eight semaphores.

Caution

The default `vba_univ` adapter configuration maps UNIVERSE II CSRs to the VMEbus for VMEbus backplane (`vb`) network driver use. Other drivers should not access the CSRs on the VMEbus except with extreme caution, because register changes may affect adapter code.

The default configuration of the UNIVERSE II CSRs window on the VMEbus is as follows:

CSR window - enabled	VMEbus base address 0xFFFF0000, 4KB, A32 supervisory data/program

You determine where in VMEbus space the UNIVERSE II CSRs are configured by modifying the following `vba_univ` subsystem attributes:

CSR window enabled or disabled (Section 2.3.5.1)
VMEbus base address (4 KB aligned) (Section 2.3.5.2)
VMEbus address modifiers (Section 2.3.5.3)

2.3.5.1 Enabling or Disabling the CSR Window

You can specify one of the following values to the CSR_Ena attribute to enable or disable the CSR window. This value is stored in the VMEbus Register Access Control Register (VRAI_CTL).

0x0	CSR window is disabled
0x1	CSR window is enabled (default)

2.3.5.2 Specifying a CSR Window VMEbus Base Address

To establish the VMEbus base address for the CSR window, you specify a hexadecimal address value to the CSR_VME_Address attribute. The value can be in the range 0x0 to 0xFFFFFFFF, but must fall within the addressable range of the VMEbus address space (A32, A24, or A16) selected for that window. The CSR window must be configured on a 4 KB boundary.

2.3.5.3 Specifying CSR Window VMEbus Address Modifiers

You can specify one of the following values to the CSR_AM_Space attribute to select the VMEbus address space for the CSR window. This value is stored in the VMEbus Register Access Control Register (VRAI_CTL).

0x0	A16 address space
0x1	A24 address space
0x2	A32 address space (default)

You can specify one of the following values to the CSR_AM_Usr_Sprvsr attribute to select user mode, supervisory mode, or both for the CSR window. This value is stored in the VMEbus Register Access Control Register (VRAI_CTL).

0x1	User mode
0x2	Supervisory mode (default)
0x3	Both user and supervisory mode

You can specify one of the following values to the CSR_AM_Data_Prg attribute to select data access, program access, or both for the CSR window. This value is stored in the VMEbus Register Access Control Register (VRAI_CTL).

0x1	Data access
0x2	Program access
0x3	Both data and program access (default)

2.3.6 Mapping a Location Monitor Window to the VMEbus

As part of configuring the `vba_univ` kernel subsystem, you can map a 4 KB location monitor window to the VMEbus for your system. Any read/write access to this window triggers interrupts for all UNIVERSE II–based VMEbus modules mapping the window (usable for implementing a global interrupt facility).

Note

Only UNIVERSE II–based systems can access UNIVERSE II location monitors. Accesses from VIP/VIC-based or other systems will cause bus errors.

The default configuration of the location monitor window on the VMEbus is as follows:

Location monitor window - disabled	VMEbus base address 0xFFFF1000, 4KB, A32 supervisory data/program

This window cannot reside within the VME-to-PCI windows you configure.

You determine where in VMEbus space the location monitor window is configured by modifying the following `vba_univ` subsystem attributes:

Location monitor window enabled or disabled (Section 2.3.6.1)
VMEbus base address (4 KB aligned) (Section 2.3.6.2)
VMEbus address modifiers (Section 2.3.6.3)

No specific operating system support exists for the location monitor registers and interrupts. To connect to location monitor interrupts, device drivers must install interrupt service interfaces for the location monitor interrupts and enable or disable location monitor interrupts.

After the location monitor interrupts are connected, any VMEbus read or write access to the UNIVERSE II location monitor window mapped to the VMEbus causes the appropriate location monitor interrupt to be generated to all interrupt-connected modules.

Device drivers must reference the location monitor window specifying matching VMEbus base address and modifiers. The device driver is

responsible for knowing the location monitor window's VMEbus base address and VMEbus address modifiers.

2.3.6.1 Enabling or Disabling the Location Monitor Window

You can specify one of the following values to the `LM_Ena` attribute to enable or disable the location monitor window. This value is stored in the Location Monitor Control Register (LM_CTL).

0x0	Location monitor window is disabled (default)
0x1	Location monitor window is enabled

2.3.6.2 Specifying a Location Monitor Window VMEbus Base Address

To establish the VMEbus base address for the location monitor window, you specify a hexadecimal address value to the `LM_VME_Address` attribute. The value can be in the range 0x0 to 0xFFFFFFFF, but must fall within the addressable range of the VMEbus address space (A32, A24, or A16) selected for that window. The location monitor window must be configured on a 4 KB boundary.

2.3.6.3 Specifying Location Monitor Window VMEbus Address Modifiers

You can specify one of the following values to the `LM_AM_Space` attribute to select the VMEbus address space for the location monitor window. This value is stored in the Location Monitor Control Register (LM_CTL).

0x0	A16 address space
0x1	A24 address space
0x2	A32 address space (default)

You can specify one of the following values to the `LM_AM_Usr_Sprvsr` attribute to select user mode, supervisory mode, or both for the location monitor window. This value is stored in the Location Monitor Control Register (LM_CTL).

0x1	User mode
0x2	Supervisory mode (default)
0x3	Both user and supervisory mode

You can specify one of the following values to the `LM_AM_Data_Prg` attribute to select data access, program access, or both for the location monitor window. This value is stored in the Location Monitor Control Register (LM_CTL).

0x1	Data access
0x2	Program access
0x3	Both data and program access (default)

2.3.7 Configuring VMEbus Interrupts

This section addresses VMEbus interrupt request levels and how to specify VMEbus interrupt parameters to the software.

2.3.7.1 VMEbus Interrupt Request Levels

Table 2–6 lists the system priority levels (SPLs) at which VMEbus and VMEbus adapter interrupt requests are delivered to the operating system and device drivers. You can query your system's VMEbus SPLs at run time by issuing the command `sysconfig -q vba_univ`.

Table 2–6: UNIVERSE II VMEbus Interrupt Request Levels

Interrupt Request Name	Alpha VME SBC SPLs
VMEbus IRQ 1	SPLDEVLOW
VMEbus IRQ 2	SPLDEVLOW
VMEbus IRQ 3	SPLDEVLOW
VMEbus IRQ 4	SPLDEVHIGH
VMEbus IRQ 5	SPLDEVHIGH
VMEbus IRQ 6	SPLDEVHIGH
VMEbus IRQ 7	SPLDEVRT
VMEbus Reset	SPLDEVRT
Module Switches	SPLDEVRT
Location Monitors	SPLDEVRT
Adapter Errors	SPLDEVRT
VMEbus IACK	SPLDEVLOW
DMA Status	SPLDEVRT

Alpha VME SBCs do not support autovector requests.

As Table 2–6 indicates, Alpha VME SBCs generate interrupt requests that higher-level interrupt requests can preempt.

On the Alpha VME SBCs, device drivers must use the `rt_post_callout` routine for interrupts delivered at SPLDEVRT. Interrupt requests for which

this is needed are VMEbus IRQ7, any of the four module switch interrupts, and any of the four location monitor interrupts.

2.3.7.2 Setting VMEbus Interrupt Vector Parameters

You specify vectors and interrupt requests (IRQs) for a device driver using the `Vector` and `Bus_Priority` fields of a `VBA_Option` entry in the `/etc/sysconfigtab` file or in a `sysconfigtab` file fragment.

Device drivers are passed this information in the `controller` structure elements `ivnum` and `bus_priority`.

VMEbus interrupt vectors 24 to 255 are available to device drivers. Vectors 0 to 23 are reserved by the VMEbus adapter. When you specify a vector to the `Vector` field of `VBA_Option`, you must also use the `Bus_Priority` field to specify an IRQ. Valid IRQ specifications are values 1 through 7. These values correspond to VMEbus levels IRQ1 through IRQ7.

See the Autoconfiguration Support section of *Writing VMEbus Device Drivers* (available in the Device Driver Kit) for an example of adding and enabling VMEbus interrupts. See the `vme_handler_info` structure in *Writing VMEbus Device Drivers* for interrupt handler information.

2.3.7.3 Specifying Module Switch Interrupt Vectors

Specify one of the following vectors in the `Vector` field of `VBA_Option` to select the module switch interrupt you want. Use the `Bus_Priority` field to specify 7 as the IRQ level.

Module switch 0 Vector 0x1140 [CSR offset 0x348]

Module switch 1 Vector 0x1150 [CSR offset 0x34C] (default)

Module switch 2 Vector 0x1160 [CSR offset 0x350]

Module switch 3 Vector 0x1170 [CSR offset 0x354]

Module switch interrupt vectors allow a module to issue an interrupt to itself or to another module. The autoconfiguration software provides control and status registers (CSRs) for use in module switch interrupts. You can specify two CSRs in a `VBA_Option` entry in the `/etc/sysconfigtab` file or in a `sysconfigtab` file fragment. At boot time, the system searches for the specified CSRs.

The autoconfiguration software performs the appropriate bus mapping and provides `io_handle_t` values in the `addr` and `addr2` members of the driver's `controller` structure. The `addr` argument is passed to the driver's `probe` routine, while the `addr2` value must be obtained from the `addr2` member of the `controller` structure.

For example, the following VBA_Option entry specifies an A32 window address as the CSR base address. The module switch 1 CSR is an offset from this A32 address.

```
VBA_Option = Csr1 - 0xFFFF0000, ..., Vector - 0x1150, Bus_Priority - 7, ...
```

The driver structure allows you to specify the size and address type for the CSRs. For example, the following members in a driver structure indicate that the first CSR has a size of 4096 (x1000) bytes and is in the A32 supervisory data address space:

```
int     addr1_size      4096
int     addr1_atype     VME_A32_SDATA
```

For more information, see the Device Driver Kit manuals *Writing Device Drivers* and *Writing VMEbus Device Drivers*, especially the sections on the addr and addr2 members of the controller structure and on the addr1_size, addr1_atype, addr2_size, and addr2_atype members of the driver structure.

In addition, you can use the vba_map_csr routine to provide module switch interrupts. After using the vba_map_csr routine to create an I/O handle, you write to an address derived from the base address plus an offset. The following code fragment shows how the I/O handle is created:

```
io_handle_t     ioh;                /* Define type of ioh */
vme_addr_t      VME_base=0xFFFF0000; /* Base CSR address */
ioh = vba_map_csr(ctlr, VME_base, 4096,
                 (VME_A32_SDATA));
```

The following code fragment shows how the module switch interrupts are issued:

```
write_io_port(ioh+0x34C, 1, 0, data)  /* Write to CSR base address
                                      plus the offset to cause
                                      module switch 1 interrupt */
mb();
```

2.3.7.4 Specifying Location Monitor Interrupt Vectors

The location monitor interrupt vectors are as follows:

Location monitor 0	Vector 0x1100
Location monitor 1	Vector 0x1110
Location monitor 2	Vector 0x1120
Location monitor 3	Vector 0x1130

No specific operating system support exists for the location monitor registers and interrupts. To connect to location monitor interrupts, device drivers must install interrupt service interfaces for the location monitor interrupts and enable or disable location monitor interrupts.

When location monitor interrupts are connected, any VMEbus read or write access to the UNIVERSE II location monitor window mapped to the VMEbus causes the appropriate location monitor interrupt to be generated to all interrupt-connected modules.

For more information about configuring the UNIVERSE II location monitor window, see Section 2.3.6.

2.3.8 Using VMEbus Software Byte Swapping

Alpha processors are little endian, while VMEbus is big endian. The default operation of the UNIVERSE II adapter causes the transfer of bytes between Alpha processors and VMEbus to be arranged correctly. If, however, a 16-bit or 32-bit number is needed in a VMEbus register, the default operation rearranges the bytes within the transfer such that the bytes are reversed in significance.

For UNIVERSE II–based Alpha VME systems, software byte swapping must be used to handle these situations. (By contrast, VIP/VIC-based Alpha VME systems use hardware byte-swapping modes.)

For VMEbus device drivers, the Device Driver Kit (DDK) provides a VMEbus example device driver, DMAEX, and accompanying user code that offers a model for how you can implement software byte swapping. You can obtain VMEbus driver-writing documentation by purchasing a DDK, or you can browse a subset of DDK materials at the Tru64 UNIX Publications web site, currently located at the following URL:

http://www.tru64unix.compaq.com/docs/

Be sure to check for the latest DDK technical updates at the same location.

If your VMEbus device driver code must be portable across both VIP/VIC-based and UNIVERSE II–based Alpha VME systems, you can code the driver to use hardware or software byte swapping according to the system type.

2.3.9 Sharing Memory Between Big Endian and Little Endian Processors

In a shared memory environment, where packed data structures in common memory are shared between an Alpha processor (little endian) and a big endian processor, software byte swapping is required to arrange bytes properly for 16- or 32-bit quantities (such as 16-bit counter values or 32-bit VMEbus address values).

The following combination is recommended: UNIVERSE II default operation with software byte swapping on nonbyte data for the Alpha processor, and no swapping on the big endian processor.

You could implement software swapping with read/write macros that perform the swap with the following code. The purpose here is to provide code that would run on both little endian and big endian machines that have shared memory.

```
#define read_word/long(iohandle,data)                    /
        data = read_io_port(iohandle,sizeof(word/long),0);/
#ifdef LITTLEENDIAN                                      /
        swap_xx(data);                                   /
#else /* BIGENDIAN */                                   /
#endif
#define write_word/long(iohandle,data)                   /
#ifdef LITTLEENDIAN                                      /
        swap_xx(data);                                   /
#else /* BIGENDIAN */                                   /
        write_io_port(iohandle,sizeof(word/long),0,data); /
#endif
```

2.3.10 Performing VMEbus Slave Block Transfers

Alpha VME platforms are configured during adapter initialization to accept slave block transfers (SBLTs) with data widths of D08, D16, D32, or D64. After the SBC has mapped its memory onto the VMEbus by using the `dma_map_alloc` and `dma_map_load` routines, no other user interaction is needed. For information on calling the `dma_map_alloc` and `dma_map_load` routines, see the corresponding reference pages in the Device Driver Kit (available separately from the base operating system).

Memory must be mapped to the VMEbus prior to D64 slave access.

Access to memory must coincide with the configured access mode. By default, all access is allowed (supervisory and user, program and data). You can constrain access by modifying the default window mappings. See Section 2.3.4 for more information about configuring VME-to-PCI address spaces.

2.3.11 Performing VMEbus Master Block Transfers with Local DMA

The VMEbus interfaces for Alpha VME platforms provide a block-mode DMA engine. This DMA engine is capable of transferring up to 16 MB minus 2 KB of data without processor intervention, in VMEbus data widths of D08, D16, D32, or D64.

The DMA engine transfers data from the VMEbus to system memory (read) or from system memory to the VMEbus (write). The hardware interface handles the segmentation of the transfer. This ensures that the VMEbus

specification is not violated in relation to crossing VMEbus 256-byte boundaries for D16 and D32 or 2 KB boundaries for D64.

The DMA engine is configured to give up the VMEbus during the transfer and to rearbitrate for the VMEbus again to continue the DMA. The time between when the DMA engine gives up the bus and rearbitrates for the bus is called the **interleave** period. During the interleave period, single-cycle VMEbus cycles, receipt of slave block transfers (SBLTs), or other operations may be performed.

The master block transfer (MBLT) hardware interface presents address modifiers of user block or supervisory block to the VMEbus, based on parameters passed in the software programming interface. The device or system on the VMEbus must be able to interpret these address modifiers; otherwise, bus errors may occur.

You can use the MBLT hardware interface to:

- Transfer data to and from those VMEbus devices that do not have their own DMA engine
- Move data between VMEbus device memory and system memory
- Transfer data to and from other systems that have their memory mapped to the VMEbus

The MBLT hardware interface supports DMA block-mode transfers to and from VMEbus A24 and A32 address space only.

2.3.11.1 Routines for Master Block-Mode Transfers

To use master block transfers (MBLTs) with the local hardware DMA engine, you must invoke the following routines and supply specific flag values:

```
vba_set_dma_addr
dma_map_alloc
dma_map_load
vba_dma
dma_map_unload
dma_map_dealloc
```

For information on calling these routines, see the corresponding reference pages in the Device Driver Kit (available separately from the base operating system).

The flag values DMA_IN and DMA_OUT have specific meaning for VMEbus support with respect to the `dma_map_alloc`, `dma_map_load`, and `vba_dma` routines. These flags indicate to the low-level VMEbus `dma_map_alloc`, `dma_map_load`, and `vba_dma` routines that the MBLT hardware DMA engine is to be used and the direction of the transfer.

Specifying DMA_IN implies a read from the VMEbus to system memory. Specifying DMA_OUT implies a write from system memory to the VMEbus. You use the `vba_set_dma_addr` routine to pass the flag values and the VMEbus address at which the transfer is to occur.

The VMEbus block-mode DMA engine on the VMEbus adapter is a single entity that must be shared among various device drivers. Specifying DMA_SLEEP causes the device driver to block in the `vba_dma` routine if the DMA engine is already being used. If DMA_SLEEP is not specified and the DMA engine is being used, `vba_dma` returns an error.

The following sample code shows how to invoke the MBLT hardware DMA engine for a block-mode read operation. The code uses a VMEbus transfer width of D32 to invoke a 256 KB transfer from VMEbus address A24 0x400000 to system memory. The code also allocates resources to handle transfers up to 1 MB in size. This allows `dma_map_load` and `vba_dma` to be invoked multiple times with varying size buffers. You can change the code to perform writes by substituting DMA_OUT for DMA_IN.

```
struct controller *ctlr;
vme_addr_t        vme_addr = 0x40000;
unsigned long     max_bc = (1024*1024);
unsigned long     rtn_bc;
char              *buffer;
unsigned long     buffer_bc = (1024 * 256);
sglist_t          dma_handle = (sglist_t)NULL;
vme_atype_t       flags = (VME_A24_UDATA_D32|DMA_IN|DMA_SLEEP);
int               rtn_flags;
/*
 *  Allocate a buffer (256 KB) to be used for the transfer
 */
MALLOC(buffer,(char *),buffer_bc,M_DEVBUF,M_WAITOK);
/*
 *  Specify a VMEbus address of 0x40000
 *  Specify flags
 *     A24 address space
 *     User mode
 *     Select DMA engine for a read (DMA_IN) and
 *       wait for DMA engine (DMA_SLEEP)
 */
rtn_flags = (int)vba_set_dma_addr(ctlr,flags,vme_addr);
/*
 *  Allocate DMA resources for up to 1 Mbyte transfer
 *    Specify flags returned from vba_set_dma_addr() above
 *    The return value from dma_map_alloc() should equal max_bc
 */
rtn_bc = dma_map_alloc(max_bc,ctlr,&dma_handle,rtn_flags);
/*
 *  Call dma_map_load() to load the resources for the
 *  DMA block-mode engine
 *    Specify the dma_handle returned from dma_map_alloc()
 *    Specify flags returned from vba_set_dma_addr()
 *    The return value from dma_map_load() should equal buffer_bc
 */
rtn_bc = dma_map_load(buffer_bc,
                      (vm_offset_t)buffer,
                      0,
                      ctlr,
                      &dma_handle,
```

```
                    0,
                    rtn_flags);
/*
 * Call vba_dma() to start up and monitor the VME adapter's block-mode
 *   DMA engine. Specify the dma_handle returned from dma_map_alloc.
 *   The return value from vba_dma() is the actual bytes transferred.
 *   This value should be the same as value buffer_bc. If not, then
 *   an error was detected during the transfer.
 */
rtn_bc = vba_dma(ctlr,dma_handle);
/*
 *  Unload and free DMA resources
 */
dma_map_unload(0,dma_handle)
dma_map_dealloc(dma_handle)
```

2.3.11.2 Restriction on VMEbus Master Block Transfers

The following restriction applies to using master block transfers (MBLTs) on UNIVERSE II–based Alpha VME platforms: The data buffer address and the VMEbus transfer address must be aligned exactly; that is, the 2 lowest bits must match.

For the best DMA performance, the data buffer address and the VMEbus transfer address should be word-aligned for D16, longword-aligned for D32, or quadword-aligned for D64.

2.3.12 Using the Realtime Interrupt-Handling Routine rt_post_callout

Interrupt service interfaces (ISIs) executing at SPLDEVRT (SPL 6) must not call kernel routines directly. The `rt_post_callout` routine allows the calling process to defer execution of a function until a time when kernel routines can be invoked. The function invoked by `rt_post_callout` runs at an elevated SPL and is subject to the same restrictions as an ISI.

The syntax for the function invoked by `rt_post_callout` is as follows:

```
int (*function)(),
long arg1,
long arg2 );
```

The parameters for the `rt_post_callout` routine are as follows:

`function`	Name of the function to be invoked
`arg1`	The first argument passed to the function
`arg2`	The second argument passed to the function

If `rt_post_callout` is called again with the same function and arguments specified, then the duplicate invocation is dismissed before the first invocation has executed.

The following example is for an interrupt service interface (ISI) that runs at SPLDEVRT:

```
rt_dev_intr(unit)
        int unit;
{
 register struct rt_softc *sc = rt_softc[unit];
 rt_post_callout(user_wakeup_interface,   /* User wakeup function */
                 (long) &sc->error_recovery_flag, /* Event to wake*/
                 (long) NULL);            /* Unused argument      */
 return;
}
```

The following example shows a user-written function to wake up an event called by the `rt_post_callout` routine:

```
void user_wakeup_interface ( arg1, arg2 )
long arg1;
long arg2;
{
   thread_wakeup( (vm_offset_t) arg1);
}
```

3

Configuring a VMEbus Backplane (vb) Network

This chapter explains how to set up a VMEbus backplane–based network in which Alpha VME single-board computers (SBCs) operate as Ethernet nodes.

The VMEbus backplane (vb) interface provides access to an Ethernet network through the VMEbus backplane driver, which acts as an Ethernet Datalink Layer driver. This interface allows VMEbus-based systems to communicate directly over the VMEbus to other VMEbus-based systems on the same backplane, or on other Ethernet-connected systems outside the backplane through a gateway node on the backplane.

Both the Tru64 UNIX and VxWorks for Alpha (Version 3.1 or higher) software support the vb driver as well as communication between these systems on the same backplane. The Tru64 UNIX vb driver is supported on AXPvme and Alpha VME SBCs and on Alpha VME 2100 systems.

The VMEbus backplane interface requires you to modify the /etc/sysconfigtab file on your AXPvme or Alpha VME system in order to configure the vb driver and to map VMEbus windows for the system. Mapping the VMEbus windows on one node requires knowledge about every node in the vb network.

Note

Do not modify any vme_vba kernel subsystem attributes. To configure a vb network node, you modify attributes of the vb driver (vb:) and the system's VMEbus adapter (vba_vipvic: or vba_univ:).

This chapter addresses the following topics relating to the use of the vb interface on Alpha VME systems:

- VMEbus backplane (vb) network overview (Section 3.1)
- Configuring vb network nodes (Section 3.2)
- Modifying vb driver attributes (Section 3.3)
- Modifying vba_vipvic adapter attributes (Section 3.4)
- Modifying vba_univ adapter attributes (Section 3.5)

- VIP/VIC two-node network example (Section 3.6)
- UNIVERSE II two-node network example (Section 3.7)
- Related `ioctl` commands (Section 3.8)
- Diagnostic messages (Section 3.9)
- Errors (Section 3.10)

3.1 VMEbus Backplane (vb) Network Overview

Tru64 UNIX provides a VMEbus backplane (`vb`) driver that allows systems to communicate over a VMEbus backplane using Ethernet protocols.

The backplane driver is compatible with the other parts of the network subsystem; that is, all higher-level network protocols are immediately available over the backplane, just as they are over the Ethernet. Socket communication, remote login, remote file access, NFS, and remote procedure calls are all simultaneously available to and from any processor on the backplane. Using these network facilities over the backplane is indistinguishable from using any other network medium.

By default, the `vb` driver is not configured to run when the system is booted and must be explicitly turned on for the node to participate in the backplane network.

Configuring nodes in a `vb` network can be simple or complex, depending on the specific system needs. At a minimum, you must configure the `vb` driver to be turned on and you must specify the Ethernet hardware address of the target system. By default, an unconfigured driver will not start up.

You can use all other default `vb` characteristics without change, as long as you configure the necessary system VMEbus window space correctly. You can also tailor several backplane node characteristics to meet specific system and application needs.

VMEbus addresses are used in two ways in the `vb` driver:

- To map local memory onto the VMEbus for client message queues
- To interrupt nodes on the `vb` network when data is sent

The following subsections describe how VMEbus addresses are used for client communication and for interrupting nodes on the `vb` network.

3.1.1 VMEbus Addresses Used for Client Communication

A `vb` network is made up of two or more nodes in a VMEbus backplane cage that communicate by way of local memory mapped onto the VMEbus. Nodes that participate in the `vb` network provide local memory for client message

queues. Other backplane nodes map to this memory over the VMEbus and write data to this local memory; this is what is meant by "sending" messages to a node on the backplane network.

The VMEbus has three different basic address spaces to which system VMEbus windows may be mapped: A16, A24, and A32. Each system in a VMEbus backplane must configure a client communication VMEbus window (and a mailbox-interrupt VMEbus window, discussed later) in a unique manner, such that the windows do not overlap across the backplane. See Section 2.2 (VIP/VIC-based Alpha VME systems) or Section 2.3 (UNIVERSE II–based Alpha VME systems) for more information on configuring VMEbus address spaces.

The `vb` driver uses either A24 or A32 space to map its client communication queues (data) to the VMEbus. You specify the following information regarding the queues for each backplane node:

- The address space in which to map the queues (A24 or A32) as well as other address space modifiers (supervisory/user, program/data)
- The address within A24 or A32 space at which the queues will be mapped to the VMEbus, specified as an offset from the base of the queues' chosen system VMEbus window
- The total size of the area needed to map the communication queues

Default values are defined for these items, but you can reconfigure your `vb` and VMEbus characteristics by adding or modifying values in the `/etc/sysconfigtab` file, as described in Section 3.2.

Whatever you configure the values to be, you must modify the client communication window to accommodate the chosen values. The window (A24 or A32) base and size specified must be unique across the backplane, and its size must be big enough to fit the queue size specified, starting at the offset specified.

You can configure VMEbus windows on a per-system basis by adding or modifying values for each window's VMEbus base address and size in the `/etc/sysconfigtab` file. See Section 2.2 (VIP/VIC-based Alpha VME systems) or Section 2.3 (UNIVERSE II–based Alpha VME systems) for more information on modifying the base address and size of VMEbus windows.

Note

If you do not uniquely configure the client communication VMEbus windows for the backplane nodes on the `vb` network, unpredictable behavior may occur. An error message similar to the following prints to the console of a node whose client communication VMEbus window overlaps that of a node that has mapped the window and is actively communicating through it, even if it is with a device other than the `vb` driver:

```
vba0 errors_inter: VIP/VIC errors detected
        VIC BESR 0x50 - VIP BESR 0x40400 VIC DMASR 0x8
        VMEbus timeout
        local bus error - LBERR* asserted to VIC
        Inbound error - invalid s/g or VMEbus slave access error
```

3.1.2 VMEbus Addresses Used for Interrupting

Module-switch (mailbox-interrupt) settings regulate interrupt activity in the `vb` backplane network. When a node sends data to another node, the sending node generates an interrupt on the receiving node by using module switches.

An interrupt is generated by writing to a particular offset from the base of a mailbox-interrupt window, which is an A16 (VIP/VIC) or A16/A24/A32 (UNIVERSE II) VMEbus window on the node to be interrupted. The offset determines the particular module switch to use for interrupting a node.

You must configure each node's mailbox-interrupt window to be unique across the nodes in the VMEbus backplane. You can configure VMEbus windows on a per-system basis by adding or modifying values for each window's VMEbus base address and associated attributes in the `/etc/sysconfigtab` file. See Section 2.2 (VIP/VIC-based Alpha VME systems) or Section 2.3 (UNIVERSE II–based Alpha VME systems) for more information on modifying the base address and size of VMEbus windows.

Four module switches are associated with each node's mailbox-interrupt window. You specify the module switch to use for interrupting by adding or modifying values for the following driver attributes in `/etc/sysconfigtab`:

- A module-switch offset value in `VB_Mailbox_Offset`
- A module-switch vector number in the `Vector` field of the `VBA_Option` entry
- For UNIVERSE II–based Alpha VME systems, VMEbus address modifiers for the mailbox-interrupt window in `VB_Mailbox_Addr_Type`

Additionally, you must verify that the `VB_Interrupt_Interface` attribute is set to 1 to select interrupt mode over polling mode.

If you prefer, you can use the default module-switch offset and vector values, which select module switch 1. Adapter-specific offset and vector values are listed in Section 3.3.

Whatever you configure the values to be, you must modify the A16 (VIP/VIC) or A16/A24/A32 (UNIVERSE II) mailbox-interrupt window base address to be unique among the nodes in the VMEbus backplane for interrupting to work. (Only one node in the backplane can use the default mailbox-interrupt window base address.)

However, you can configure the module switch used to interrupt a particular node individually on a per-node basis (not necessarily uniquely).

3.1.3 Box Manager Node

Because a `vb` network is made up of two or more nodes in a VMEbus backplane cage that communicate via local memory mapped onto the VMEbus, information about which nodes are participating in the network must be stored so that all nodes can access this information. The information is stored in the local memory of a single backplane node, called the **box manager**.

The box manager node is a special client in that it maps this global information onto the VMEbus in addition to mapping its client communication queues.

The box manager maps the global information onto the VMEbus at an address that is known to all other nodes in the backplane network (the well-known address). When non-box-manager nodes boot, they read information from the well-known address to see what other nodes are in the network. The well-known address must reside in the particular system VMEbus window (A24 or A32) with modifiers (supervisory/user, program/data) that are also well known to other nodes in the `vb` network. The combination of the address and its modifier uniquely specifies where the box manager global data resides on the VMEbus for all nodes to see.

The well-known address is configurable through `/etc/sysconfigtab` and defaults to 0xBC0000. The address space that it is mapped to (A24 or A32) is also configurable and defaults to A24 address space, supervisory mode, and data space. (For more information on configuring the well-known address and modifiers, see Section 3.3.2.)

The network administrator must configure only one node to be the box manager node. A node is a box manager if the well-known address is contained within the node's system VMEbus window (either A24 or A32, depending on the configured value of the box manager address modifier). No other switch or value specification is needed to identify a box manager. Note that you do not have to set the base VMEbus window address to the

well-known address; the well-known address must simply be contained within a valid VMEbus system window.

When a node boots, it determines whether or not it is the box manager node by comparing the well-known address to its configured system VMEbus window range. A node that is not the box manager node is called a client node.

The box manager node is also just another network client, and it has local communication queues mapped to the VMEbus just like any other client. The difference is in the placement of those queues mapped onto the VMEbus. The box manager has two sets of data that must be mapped to the VMEbus: the box manager global data and the client communication queues.

By default, box manager global data and client communication queues are mapped to the same address space, A24. In the default case:

- The offset of the communication queues from the base window specified in `/etc/sysconfigtab` for the queues is ignored.
- The communication queues are mapped directly following the global data starting at the well-known address.

In addition, the combined size of the global data and the communication queues is adjusted to be equal to the configured size of the communication queues (the default for which is 0x40000, or 256 KB). You do not need to deal with the size of the box manager global data when you determine what your system VMEbus window size should be for the box manager node.

However, you can configure the global data and the communication queues to be mapped to different spaces (A24 and A32). In this case, the communication queues are mapped like any other client node. They are mapped at the configured offset from the base of its configured window. The global data is mapped to the well-known address, for a size of 0x6000 bytes. You must be sure that both system windows, A24 and A32, will accommodate either the well-known address or the communication queues.

The box manager node must be the first node in the backplane to boot, so that the global memory is mapped to the well-known address before other nodes attempt to read from it.

You must boot the VMEbus system controller for the VMEbus crate (set by the appropriate jumper on the module) before any other node that is participating in the `vb` backplane and before any other node that is using the VMEbus. This is because when the system controller is booted, it can reset the VMEbus registers of all other nodes. If the VMEbus system controller is not the box manager, ensure that the system controller boots before the box manager node, or that the system controller is not booted while the `vb`

network is up and running. Note that if the the system controller is not the box manager, the system controller cannot participate in the vb network.

3.1.4 Network Participation

Nodes in a backplane network communicate via memory mapped onto the VMEbus. If this memory becomes unmapped, or if the VMEbus is reset for any reason, the mapping is no longer valid. Any read or write operations to a remote node that uses the invalid mapping could cause a panic or machine check on the system performing the read or write. To reduce the possibility of this occurring, the nodes in the vb network maintain **liveness** with the rest of the network.

To maintain liveness, when a node enters the vb network, it begins continually updating a counter in the global memory called its **heartbeat**. In addition, all nodes on the network continually check the vb heartbeat of other nodes, including the box manager node, to see if they are still alive and able to participate in the network in a timely manner.

If the heartbeat of a remote node is no longer being updated, communication to that node must stop in anticipation of the remote node's VMEbus mapping becoming invalid. For example, if a node is rebooted, its heartbeat ceases to be updated and the rest of the backplane nodes eventually lose liveness with that node and stop communicating with it.

When a node is shut down in a controlled manner (using /usr/sbin/shutdown), the vb driver notifies the other vb nodes that it is shutting down, so that they can stop communicating. If a node is shut down in an uncontrolled manner (panic or halt), the current VMEbus mappings remain valid until you reinitialize the system. This allows time for other vb network nodes to lose liveness with the node before an invalid mapping reference occurs.

After you fully reboot the shutdown node, it can reenter the vb network and be seen by the other vb network nodes again.

If node A loses liveness with node B, node B cannot reenter the vb network without rebooting. You cannot restart the vb driver without rebooting. This restriction is due to the need for the restarting node to probe the well-known address to see if a box manager memory is mapped to the well-known address. This probing is supported only during the booting stage.

Response time is an important aspect of liveness. Even if a node is not shut down, it may respond too slowly to vb network traffic to be considered alive. In these cases, it may be in the best interest of the rest of the vb network to cease communication with that node. For example, a node may have a realtime application running at a realtime priority above that of the vb network driver, and in fact higher than many system functions. Without

network traffic being processed in a timely manner, backups or message loss could occur on any node attempting to send data to the node.

The liveness feature of the drivers allows remote nodes to notice that the node's heartbeat is not being updated (because the node is devoted to the realtime application) and stop attempting to communicate with it. In addition, you could use a long liveness interval in a stable network configuration (one that does not expect frequent shutdowns) to allow a light load on the vb network to continue in the midst of expectedly high realtime priority usage.

3.2 Configuring vb Network Nodes

To configure a vb network node, you perform the following steps:

1. Examine the default or current configuration attributes of the vb driver (vb:) and of the system's VMEbus adapter (vba_vipvic: or vba_univ:).

   If an existing vba_vipvic: or vba_univ: entry in /etc/sysconfigtab indicates that adapter defaults have already been modified for other VMEbus device drivers in the system, you must factor the needs of other drivers into any changes you make for the vb driver.

2. As needed, modify the /etc/sysconfigtab file to add or modify values for vb driver and VMEbus adapter attributes. You must turn on the vb driver and you must specify the node's Ethernet hardware address. Also, as part of modifying VMEbus adapter attributes, you need to configure each node's VMEbus system windows with the other participating nodes' VMEbus window configurations in mind. Sections that follow describe these tasks in detail.

   Note

   Do not directly edit /etc/sysconfigtab. Instead, use the sysconfigdb facility, as described in the sysconfigdb(8) reference page. It is recommended that you maintain private sysconfigtab file fragments for vb and VMEbus adapter attributes and use sysconfigdb switches to add (-a -f), delete (-d), or merge (-m -f) attribute values for a particular subsystem. The examples in Section 3.6 and Section 3.7 illustrate this approach.

3. Reboot the vb node. You must always reboot after modifying driver or adapter subsystem attributes.

4. When a configured vb node boots, you must use the netsetup command to register the vb driver as a new network driver. Assign each vb node a

unique IP address that is a subnet used exclusively by the vb network, to differentiate between the Ethernet network and the vb network. The participating nodes must be specified in the /etc/hosts file. For information on setting up a new network, see *Network Administration: Connections* and *Network Administration: Services*.

You must configure and boot the box manager node before configuring and booting any other nodes. Also, if the box manager is not the VMEbus system controller, the VMEbus system controller module must boot before the box manager. Otherwise, when the system controller is booted, it may reset the entire VMEbus backplane network.

When you boot each configured node, the VMEbus backplane driver becomes available. During the boot, the console displays diagnostic messages prefixed with the string VB:. The box manager displays the following message at startup:

```
VB: This is the box manager node
```

A client (non-box-manager) node displays the following message at startup:

```
VB: Box mgr address space is not configured for this system,
    thus this node is not the box manager node (OK).  Be sure
    that there is a box manager in the network.
```

Caution

Make sure that only one node comes up as the box manager. If more than one node comes up as the box manager, it means that the system VMEbus address window has been configured to contain the well-known address (whose default is 0xBC0000) on more than one node. This results in unpredictable behavior and, at a minimum, causes the vb network to fail.

3.3 Modifying vb Driver Attributes

The vb driver attributes are configurable on a per-node or per-vb-network basis, as described in detail in this section.

First you examine the default or current configuration attributes of the vb driver and the system's VMEbus adapter. Table 3–1 lists the default values for vb driver parameters.

Table 3–1: VMEbus Backplane (vb) Network Driver Defaults

Parameter	Default	Meaning
Per-node vb attributes:		
Module_Config_Name	vb	Driver name is vb
VB_Startup_State	0	Driver is off
VB_Client_Addr_Type	0x7	Client communication window address modifiers: A24 space, supervisory mode, and data access
VB_Client_Vme_Window_Size	0x40000	Size of communication queues area is 256 KB
VB_Client_Vme_Win-dow_Offset	0x0	Map client queues at offset 0x0 from the client communication window base address
VB_Interrupt_Interface	1	Message response is interrupt driven
VB_Liveness_Timeout	10000	Remote liveness tests are 10000 milliseconds (10 seconds) apart
VB_Mailbox_Addr_Type	0xE	(UNIVERSE II only) Client mailbox-interrupt window address modifiers: A16 space, supervisory mode, and data access
VB_Mailbox_Offset	0x23 or 0x34C	Module switch 1 is selected by offset 0x23 (VIP/VIC) or 0x34C (UNIVERSE II)
VB_Maxnodes	10	Maximum nodes allowed in the vb network is 10
VB_Netid	—	Ethernet hardware address of the node must be supplied for the network to start up
VB_Give_Up	1	Time out if the VB_Probe_Period is exceeded
VB_Probe_Period	1	Number of minutes to probe the box manager's well-known address before exiting driver is 1
VB_Census_Change	0	Do not display node mapping state changes
VB_Transfer_Type	0	Use only programmed I/O (PIO) transfers over the bus
VB_DMA_Threshold	256	If VB_Transfer_Type equals 1, transfers equal to or exceeding 256 bytes will use direct memory access (DMA) rather than PIO

Table 3–1: VMEbus Backplane (vb) Network Driver Defaults (cont.)

Parameter	Default	Meaning
VB_DMA_Dwidth	0	If VB_Transfer_Type equals 1, the D16 data width will be used for DMA transfers over the bus
Per-network vb attributes:		
VB_Box_Mgr_WK_Addr	0xBC0000	Box manager's well-known VMEbus address is 0xBC0000 (must match on every node)
VB_Box_Mgr_WK_Addr_Type	0x7	Box manager global data address modifiers: A24 space, supervisory mode, and data access (must match on every node)
VB_Maxmtu	1500	Maximum transfer unit (mtu) size is 1500 bytes (configurable on the box manager node only)

Table 2–1 and Table 2–4 list the parameter defaults for the VIP/VIC and UNIVERSE II VMEbus adapters, respectively.

If the existing vb: entry in /etc/sysconfigtab indicates that vb driver defaults have already been modified, you may need to factor the previous changes into your new changes. If an existing vba_vipvic: or vba_univ: entry in /etc/sysconfigtab indicates that adapter defaults have already been modified for other VMEbus device drivers in the system, you must factor the needs of other drivers into any changes you make for the vb driver.

If you wish to change a vb driver attribute from its default or current value, you enter the attribute and its new value after the label vb: in the /etc/sysconfigtab file or in a sysconfigtab file fragment.

Note

Do not directly edit /etc/sysconfigtab. Instead, use the sysconfigdb facility, as described in the sysconfigdb(8) reference page. It is recommended that you maintain a private sysconfigtab file fragment for vb attributes and use sysconfigdb switches to add (-a -f), delete (-d), or merge (-m -f) vb attribute values. The examples in Section 3.6 and Section 3.7 illustrate this approach. You must always reboot after changing vb driver attributes.

You must also add or modify vba_vipvic: or vba_univ: adapter attribute values to map unique VMEbus windows for client communication and mailbox interrupts, as described in Section 3.4 and Section 3.5, respectively.

The following code example shows a sample vb: entry in the /etc/sysconfigtab file or in a sysconfigtab file fragment, including the associated VBA_Option bus configuration structure. Line breaks have been added to the VBA_Option entry for clarity.

In this example, only the VB_Startup_State and VB_Netid parameters have been modified from their defaults. These modifications enable the vb driver to start up and participate in a vb network. After you complete your vb driver and VMEbus adapter modifications, you must reboot the system.

```
vb:
#
# %%%VB
#
   VB_Startup_State = 1
   VB_Netid = 08-00-2b-e2-48-48
   VBA_Option = Manufact_Name - 'Digital',
                Product_Name - 'VME Backplane Network Driver',
                Bus_Instance - 0, Driver_Name - vb, Driver_Instance - 0,
                Csr1 - 0, Csr2 - 0, Vector - 0x1150, Bus_Priority - 7,
                Type - C, Adpt_Config - N
```

3.3.1 Modifying Per-Node vb Attributes

The following vb driver attributes are configurable on a per-node basis in sysconfigtab; values can differ on each node:

- Module_Config_Name

  Specifies the driver name as an unquoted ASCII string. The default value is vb.

- VB_Startup_State

  Specifies the startup state of this driver. The default value is 0 (off). You must change this value to 1 (on) to start up the vb network.

- VB_Client_Addr_Type

  Specifies address modifiers for the VMEbus address space used for the client node's message queues. You must specify the numerical equivalent of the desired set of address modifier (AM_) flags, among the following: AM_A24 (0x1), AM_SUPER (0x2), and AM_DATA (0x4).

  You can map a node's client communication queue memory to the VMEbus in either the A24 or A32 address space (AM_A24 set or clear), either in supervisory mode or user mode (AM_SUPER set or clear), and either in data or program space (AM_DATA set or clear). The default is AM_A24|AM_SUPER|AM_DATA, which equals 0x7. A32 program space in user mode would be represented as 0x0 (no flags set).

- VB_Client_Vme_Window_Size

  Specifies the size (in bytes) of an area within the client communication window (as characterized by VB_Client_Addr_Type) to be used by

the vb driver for its communication queues. The bigger the size, the greater the number of message packets that will be preallocated for communication.

The default size is 0x40000 (256 KB). If the maximum transfer unit (VB_Maxmtu) and maximum nodes (VB_Maxnodes) parameters, described below, are left at their default values of 1500 bytes and 10 nodes, the 256 KB window size is enough for approximately 150 packets to be reserved for the queues. With a maximum of 10 nodes in the network, this default allows for approximately 15 packets per node to be devoted exclusively to communication between the local node and each of the other nodes. Increasing VB_Maxmtu would decrease the number of packets available per node.

- VB_Client_Vme_Window_Offset

  Specifies the offset from the client communication window base address (A24 or A32, as characterized by VB_Client_Addr_Type) at which to map client queues for other nodes to see. The default is 0x0, which maps the queues at the beginning of the base address.

  You must be able to adjust queue mappings because, if other VMEbus drivers in the system map memory to specific VMEbus addresses, there may be conflicts. In the event of a conflict, you can either adjust a system VMEbus window base address or modify the offset value such that the queues start at a different VMEbus address.

  Although the default offset of 0x0 works well, you should consider changing the offset to a value equal to the A24 or A32 window size minus the size of the client communication queues (VB_Client_Vme_Window_Size defaults to 256 KB, 0x40000). For example, in a 2 MB (0x200000) A24 window, specify an offset of 0x1C0000. This moves the client communication queues to the top of the window, which reduces fragmentation within the window and minimizes potential conflict with the memory needs of other VMEbus drivers.

  If you change the offset, make sure the value is on a page boundary (0x2000 bytes).

- VB_Interrupt_Interface

  Specifies an interface for determining whether messages have been sent to the vb driver's queues: interrupt (1) or polled (0). You should use the default value of 1 for better performance. The vb driver uses module switch interrupts.

  If you use the interrupt interface, you must ensure that the base address for the VMEbus window that maps the inbound mailbox interrupts is unique among the nodes in the backplane, as configured in sysconfigtab.

- VB_Liveness_Timeout

Specifies the interval in milliseconds between remote node liveness tests. By default, a node checks whether a remote node is still alive every 10000 milliseconds (10 seconds).

Be careful if you modify this value. An interval that is too short could cause nodes to lose liveness with each other too easily, and a lost node must be rebooted to resume communication. An interval that is too long (or 0, which specifies no liveness checking) could cause delays in determining that a remote node has gone down. The node could attempt to communicate with a shut-down node after the VMEbus mapping is no longer valid.

- `VB_Mailbox_Addr_Type` (UNIVERSE II only)

  For UNIVERSE II–based Alpha VME systems only, specifies address modifiers for the VMEbus address space used to map the client node's inbound mailbox interrupts. You must specify the numerical equivalent of the desired set of address modifier (AM_) flags, among the following: AM_A24 (0x1), AM_SUPER (0x2), AM_DATA (0x4), and AM_A16 (0x8).

  On UNIVERSE II–based nodes, you can map a node's mailbox interrupts to the VMEbus in A16, A24, or A32 address space. Specifying AM_A16 set and AM_A24 clear selects A16; specifying AM_A24 set and AM_A16 clear selects A24; and specifying both AM_A16 and AM_A24 clear selects A32. You also can map the mailbox interrupts either in supervisory mode or user mode (AM_SUPER set or clear), and either in data or program space (AM_DATA set or clear). The default is AM_A16|AM_SUPER|AM_DATA, which equals 0xE. A32 program space in user mode would be represented as 0x0 (no flags set).

  For UNIVERSE II–based nodes, the VMEbus address modifiers you specify for this attribute must match the adapter's CSR window attributes. See the descriptions of the `CSR_AM_Space`, `CSR_AM_Usr_Sprvsr`, and `CSR_AM_Data_Prg` attributes in Section 2.3.

  For VIP/VIC-based Alpha VME systems, do not specify this attribute; the VMEbus window that maps inbound mailbox interrupts is always A16 data space in supervisory mode.

- `VB_Mailbox_Offset`

  Selects a mailbox for inbound interrupts by specifying an offset from the mailbox-interrupt window base address.

  You use module switches to create `vb` driver interrupts on the backplane. You can use any of four module switches for interrupts in each mailbox-interrupt window. For each module switch, you must specify a particular offset value for `VB_Mailbox_Offset` and specify a particular vector number in the `Vector` field of the `VBA_Option` entry.

  For VIP/VIC-based Alpha VME systems, the offset and vector values are:

Module switch 0	A16 offset 0x21, VBA_Option vector 0x1140
Module switch 1	A16 offset 0x23, VBA_Option vector 0x1150 (default)
Module switch 2	A16 offset 0x25, VBA_Option vector 0x1160
Module switch 3	A16 offset 0x27, VBA_Option vector 0x1170

The default is module switch 1. Remote nodes can use offset 0x23 added to a target node's mailbox-interrupt window base address (see examples in Section 2.2.4.4 and Section 3.6) to cause an interrupt on the target node when the vb driver writes to the address.

For UNIVERSE II–based Alpha VME systems, the offset and vector values are:

Module switch 0	Offset 0x348, VBA_Option vector 0x1140
Module switch 1	Offset 0x34C, VBA_Option vector 0x1150 (default)
Module switch 2	Offset 0x350, VBA_Option vector 0x1160
Module switch 3	Offset 0x354, VBA_Option vector 0x1170

The default is module switch 1. Remote nodes can use offset 0x34C added to a target node's mailbox-interrupt window base address (see examples in Section 2.3.7.3 and Section 3.7) to cause an interrupt on the target node when the vb driver writes to the address.

The mailbox-interrupt window base address must be unique among all nodes in the backplane. However, the offset need not be unique.

If you change the module switch from the default of 1, this change must be reflected in both the VB_Mailbox_Offset attribute and the Vector field of the VBA_Option entry for interrupts to work on the system.

- VB_Maxnodes

  Specifies the maximum number of nodes allowed in the vb network. The default value is 10. The maximum you specify cannot exceed 32. This value is examined by the vb box manager only, and determines the maximum number of nodes that may enter the vb network while the box manager is booted.

  All other client nodes adjust their maximum-nodes value according to the box manager's value and do not have to know the box manager's value ahead of time.

- VB_Netid

  Specifies the Ethernet hardware address of the node as an unquoted ASCII string; for example, 08-00-2b-e2-48-48. You must fill in this field with the correct Ethernet hardware address. The vb network address is derived from the unique Ethernet hardware address and is the shadow

Ethernet address. If this value is not filled in, the vb driver does not start up and an error message is displayed.

One way to obtain the Ethernet hardware address of a running system is `netstat -I ln0` (or `tu0` or other Ethernet device). You can also obtain the Ethernet address at the console prompt of a nonbooted system as follows:

```
>>> show dev
```

- `VB_Give_Up`

  Specifies whether the vb driver's probing of the box manager's well-known address should time out after the number of minutes specified in `VB_Probe_Period` or continue until the box manager comes up. The default is to time out (1). You can modify the value to continue probing indefinitely (0).

- `VB_Probe_Period`

  Specifies the number of minutes to probe the box manager's well-known address before timing out and exiting the driver. The default value is 1 minute. This value is ignored if `VB_Give_Up` is set to 0.

- `VB_Census_Change`

  Specifies whether to display information whenever the driver maps to a new node or unmaps from a node. The default is not to display state changes (0). If the vb driver starts up with this value set to 1, you can track state changes beginning at startup.

- `VB_Transfer_Type` *(requires Tru64 UNIX Version 5.0A or higher)*

  Specifies whether transfers over the bus use only programmed I/O (0) or can select between programmed I/O and direct memory access based on the transfer size (1). The default is 0, programmed IO only.

  If you set `VB_Transfer_Type` to 1, direct memory access (DMA) transfers will be performed whenever the transfer size equals or exceeds the value of `VB_DMA_Threshold` (256 by default); for smaller transfer sizes, programmed I/O (PIO) transfers will be performed.

  You can produce significant performance gains by allowing DMA transfers over the bus, particularly if you select the D64 data width with the `VB_DMA_Dwidth` parameter, described below. Furthermore, if all vb nodes in your network are running Tru64 UNIX 5.0A or higher, you potentially can realize even greater performance gains by modifying the per-network parameter `VB_Maxmtu`, which is described in Section 3.3.2.

  Before you enable vb DMA transfers on a node, you should consider the potential impact on your system, such as increased contention for DMA between the vb driver and devices in the system.

- `VB_DMA_Threshold` *(requires Tru64 UNIX Version 5.0A or higher)*

If DMA transfers over the bus are enabled (VB_Transfer_Type equals 1), this parameter defines the threshold (a transfer size, in bytes) at which DMA transfers are used. Whenever a transfer size equals or exceeds VB_DMA_Threshold (256 bytes by default), DMA is used for the transfer; otherwise PIO is used.

- VB_DMA_Dwidth *(requires Tru64 UNIX Version 5.0A or higher)*

  If DMA transfers over the bus are enabled (VB_Transfer_Type equals 1), this parameter defines the data width to be used for the DMA transfers. The value 0 (the default) selects D16, 1 selects D32, and 2 selects D64. If DMA is enabled, you can realize the maximum performance gain by selecting the D64 data width.

- VB_Developer_Debug

  Reserved for future use

3.3.2 Modifying Per-Network vb Attributes

The following vb driver attributes are configurable on a per-network basis in sysconfigtab; values must match exactly on every node that participates in the vb network:

- VB_Box_Mgr_WK_Addr

  Specifies the well-known VMEbus address of the box manager, to which the box manager maps 256 KB of global VMEbus data. This address and its associated 256 KB size must fit within the adapter's configured inbound VMEbus address space. The default is 0xBC0000. Be careful when modifying this value, as it must match on every node in the vb network for communication to occur.

Note

The VB_Box_Mgr_WK_Addr default of 0xBC0000 differs from the default used in previous vb driver versions, 0xA40000. The new default places the vb box manager well-known address near the top of what is assumed to be a 2 MB A24 or A32 window: 0xC00000 (window top) minus 0x040000 (256 KB size) equals 0xBC0000. This reduces fragmentation within the window and minimizes potential conflict with other VMEbus drivers on the same node allocating memory in the same window.

UNIVERSE II Restriction

Section 3.7, UNIVERSE II Two-Node Network Example, uses the value 0xFC0000 for VB_Box_Mgr_WK_Addr. This suggested value does not fit into the UNIVERSE II adapter's default special

A24/A16 outbound window, due to the allocation of the A24/A16 window's top 64 KB for A16 space. UNIVERSE II vb nodes should either adjust the box manager well-known address down by 64 KB (x10000) to 0xFB0000 to allow use of the A24/A16 window, or instead use an outbound PCI-to-VME 256 KB (or larger) window to map the 0xFC0000 value. Note that doing the latter can boost performance by allowing use of block transfers (BLTs) and a wider data path, at the cost of the added PCI resources used to map the window.

- VB_Box_Mgr_WK_Addr_Type

  Specifies address modifiers for the box manager's well-known VMEbus address. You must specify the numerical equivalent of the desired set of address modifier (AM_) flags, among the following: AM_A24 (0x1), AM_SUPER (0x2), and AM_DATA (0x4).

  You can map the box manager's global data to the VMEbus in either the A24 or A32 address space (AM_A24 set or clear), either in supervisory mode or user mode (AM_SUPER set or clear), and either in data or program space (AM_DATA set or clear). The default is AM_A24|AM_SUPER|AM_DATA, which equals 0x7. Be careful when modifying this value, as it must match on every node in the vb network. If you modify this value, make sure the address is on a page boundary (0x2000 bytes).

- VB_Maxmtu

  Specifies the maximum transmit unit (mtu) size, in bytes. Before Version 5.0A of Tru64 UNIX, this value was not configurable. Beginning with Tru64 UNIX 5.0A, *and provided all nodes in your* vb *network are running Tru64 UNIX 5.0A or higher,* you can modify this value from its default of 1500 bytes up to a maximum of 16384 (16K) bytes. (Values less than 1500 or greater than 16K default to 1500.) Specifying a larger mtu increases the size of transfer packets, resulting in fewer (but larger) packets on the transfer queues.

  You modify this value on the box manager node only; on client nodes, leave the value at its default. Client nodes obtain the mtu size from the box manager during node registration.

  Modifying VB_Maxmtu alone can produce significant performance gains in programmed I/O (PIO) transfers. However, using VB_Maxmtu in conjunction with the VB_Transfer_Type, VB_DMA_Theshold, and VB_DMA_Dwidth parameters allows you to take advantage of direct memory access (DMA) transfers over the bus and potentially realize even greater performance gains.

Note that increasing the mtu size has a significant effect on the allocation of memory resources for the complete vb network. For example, if you specify 16K as the mtu, that increase is multiplied times VB_Maxnodes, the maximum number of nodes in the system. If your system design allows, you may be able to reduce the maximum number of nodes in the system (modify VB_Maxnodes), thereby increasing the memory resources available per node.

3.4 Modifying vba_vipvic Adapter Attributes

On each node in a vb network, you must modify VMEbus adapter attributes in /etc/sysconfigtab to configure unique system VMEbus windows for client communication and mailbox interrupts. If the node is VIP/VIC-based, you add or modify values for vba_vipvic kernel subsystem attributes, such as A32_Base, A32_Size, A24_Base, A24_Size, and A16_Base. Section 2.2 describes these attributes.

Note

Do not directly edit /etc/sysconfigtab. Instead, use the sysconfigdb facility, as described in the sysconfigdb(8) reference page. It is recommended that you maintain private sysconfigtab file fragments for vba_vipvic attributes and use sysconfigdb switches to add (-a -f), delete (-d), or merge (-m -f) vba_vipvic attribute values. The example in Section 3.6 illustrates this approach.

Each system participating in the vb network must map its client communication queues to either A24 or A32 space in a unique manner. Allocated system VMEbus window space must be sufficient to accommodate the size devoted to the communication queues. In addition, the system VMEbus window of the box manager node must encompass the well-known address (default of 0xBC0000).

Although the address modifiers of the box manager well-known address and of the client communication queues are the same by default (A24/Supervisor/Data), they need not be the same. If they are not the same, configure the box manager node so that its system windows accommodate both sets of data. If they are the same, configure the box manager node so that the chosen system VMEbus window accommodates both sets of data, starting at the well-known address, for a size equal to the size of the communication queues.

For interrupting, the A16 system VMEbus window base address must also be unique for all nodes in the backplane, but the size is always 0x100.

Table 3–2 lists the VMEbus address space parameters you can modify in /etc/sysconfigtab and their defaults.

Table 3–2: VIP/VIC VMEbus Address Space Defaults

Parameter	Default	Meaning
A32_Base	0x08000000	A32 inbound DMA window base address
A32_Size	0x08000000	A32 window size (128 MB)
A24_Base	0x00C00000	A24 inbound DMA window base address
A24_Size	0x00400000	A24 window size (4 MB)
A16_Base	0x00000100	A16 interprocessor communication base address (size is always 0x100)

See the VIP/VIC Two-Node Network Example in Section 3.6 for examples of how to modify /etc/sysconfigtab for VIP/VIC-based nodes in a vb network.

Note

The size of the system VMEbus window for a node should exceed what the vb driver needs. If the vb driver uses the entire system VMEbus window, no window space remains for other VMEbus devices on the system to use.

A system administrator must carefully configure all nodes on the backplane to have large enough system VMEbus windows to accommodate the needs of each, but not so much that there is little room left for other nodes. The system administrator should make a roadmap of each system's VMEbus device addresses and sizes and fit the vb needs around the needs of the other devices, because the vb characteristics are user configurable.

3.5 Modifying vba_univ Adapter Attributes

On each node in a vb network, you must modify VMEbus adapter attributes in /etc/sysconfigtab to configure unique system VMEbus windows for client communication and mailbox interrupts. If the node is UNIVERSE II–based, you add or modify values for vba_univ kernel subsystem attributes that configure VMEbus windows. Section 2.3 describes these attributes.

Note

Do not directly edit `/etc/sysconfigtab`. Instead, use the `sysconfigdb` facility, as described in the `sysconfigdb`(8) reference page. It is recommended that you maintain private `sysconfigtab` file fragments for `vba_univ` attributes and use `sysconfigdb` switches to add (`-a -f`), delete (`-d`), or merge (`-m -f`) `vba_univ` attribute values. The example in Section 3.7 illustrates this approach.

Each system participating in the `vb` network must map its client communication queues to either A24 or A32 space in a unique manner. Allocated system VMEbus window space must be sufficient to accommodate the size devoted to the communication queues. In addition, the system VMEbus window of the box manager node must encompass the well-known address (default of 0xBC0000).

Although the address modifiers of the box manager well-known address and of the client communication queues are the same by default (A24/Supervisor/Data), they need not be the same. If they are not the same, configure the box manager node so that its system windows accommodate both sets of data. If they are the same, configure the box manager node so that the chosen system VMEbus window accommodates both sets of data, starting at the well-known address, for a size equal to the size of the communication queues.

For interrupting, you must map the node's VMEbus adapter CSRs, including mailbox-interrupt CSRs, to a VMEbus system window. On UNIVERSE II–based nodes, extra work is required to also map the VME adapter CSRs (including mailbox-interrupt CSRs) of each `vb` partner node. On UNIVERSE II–based nodes, VMEbus device CSRs are not constrained to A16/Supervisor/Data space and potentially could vary widely in address space and characteristics from node to node. For mapping purposes, you should organize the VMEbus device CSRs for all nodes in the `vb` network into a carefully designed region of VMEbus space, such that each UNIVERSE II–based node can map them using a dedicated window with consistent VMEbus attributes. You then modify `vba_univ` adapter attributes on each UNIVERSE II node to map partner-node CSRs with a dedicated window.

See the UNIVERSE II Two-Node Network Example in Section 3.7 for examples of how to modify `/etc/sysconfigtab` for UNIVERSE II–based nodes in a `vb` network.

Note

The size of the system VMEbus window for a node should be larger than what the vb driver needs. If the vb driver uses the entire system VMEbus window, no window space remains for other VMEbus devices on the system to use.

A system administrator must carefully configure all nodes on the backplane to have large enough system VMEbus windows to accommodate the needs of each, but not so much that there is little room left for other nodes. The system administrator should make a roadmap of each system's VMEbus device addresses and sizes and fit the vb needs around the needs of the other devices, because the vb characteristics are user configurable.

3.6 VIP/VIC Two-Node Network Example

The following steps show an easy way to configure two VIP/VIC nodes to run in a VMEbus backplane (vb) network: Node 0 and Node 1.

For each node, most of the VIP/VIC and vb default values listed in Table 2–1 and Table 3–1 are retained. In particular, the well-known VMEbus address of the box manager remains at its 0xBC0000 default. You should examine the attribute defaults listed in Table 2–1 and Table 3–1, invoke `sysconfigdb -l vba_vipvic` and `sysconfigdb -l vb` on each node to uncover any previous changes to those defaults, and decide which attribute values require further modification.

In this example, VIP/VIC and vb parameters that must be modified include the following:

- The A24 base address and size
- The A16 base address
- The startup state
- The node's Ethernet hardware address
- The client queues offset from the base of the node's A24 system VMEbus window

Configure the box manager node first. Make sure that it is either the VMEbus system controller node or that the system controller node is already up.

To configure Node 0, perform the following steps:

1. On Node 0, create a `sysconfigtab` file fragment in a private directory; for example, `/mypath/vipvic_sysconfigtab`. Insert the label

vba_vipvic: at the beginning of the file. In the next few steps, you will construct an indented list, immediately following the label, of the vba_vipvic attributes you wish to modify and their values.

This example assumes /etc/sysconfigtab contains no previous vba_vipvic entry. If such an entry exists, you can either remove the old entry (sysconfigdb -d) before adding the new, or merge the new attributes in with the old (sysconfigdb -m -f). You may need to factor the earlier vba_vipvic attribute values into your new modifications.

2. Change the A24 base address (vba_vipvic parameter A24_Base) from the default of 0xC00000 to something that encompasses the box manager data well-known address of 0xBC0000. For example, set the A24 base address to 0xA00000, and change the A24 size (parameter A24_Size) to 2 MB (value 0x200000), which brings the window to just below the default window address of 0xC00000. The box manager node now has an A24 window of 0xA00000 to 0xBFFFFF.
3. Change the A16 base address (parameter A16_Base) to something other than the default of 0x100; for example, 0x000.
4. The /mypath/vipvic_sysconfigtab file fragment now contains the following text:

```
vba_vipvic:
        A24_Base = 0x00A00000
        A24_Size = 0x200000
        A16_Base = 0x00000000
```

Close the file, then add its contents to /etc/sysconfigtab by issuing the following command:

```
sysconfigdb -a -f /mypath/vipvic_sysconfigtab  vba_vipvic
```

5. Create a sysconfigtab file fragment for vb attributes; for example, /mypath/vb_sysconfigtab. (If you want to use the default vb: entry provided by /etc/sysconfigtab as a starting point, you can copy the existing entry into the file fragment using the command sysconfigdb -l vb > /mypath/vb_sysconfigtab.)

   In the next few steps, you will construct an indented list, immediately following the label vb:, of the vb attributes you wish to modify and their values.
6. Change the VB startup state (vb parameter VB_Startup_State) from 0 (off) to 1 (on).
7. Specify the VB node's Ethernet hardware address (vb parameter VB_Netid). For example, if the node's Ethernet address is 08-00-26-E2-48-47, you would specify that address as an unquoted ASCII string.

8. Modify the client communication queues offset, VB_Client_Vme_Window_Offset, to map client queues at the top of the A24 window, as previously configured with the vba_vipvic attributes A24_Base and A24_Size. Mapping at the top of a window reduces window fragmentation and minimizes potential conflicts with the memory needs of other VMEbus drivers. Specify the value 0x1C0000, which equals the A24 window size (0x200000) minus the 256 KB needed for client queues (0x040000).

9. The /mypath/vb_sysconfigtab file fragment now contains the following text:

```
vb:
   VB_Startup_State = 1
   VB_Netid = 08-00-26-e2-48-47
   VB_Client_Vme_Window_Offset = 0x1C0000
```

   Close the file, then merge its contents into /etc/sysconfigtab by issuing the following sysconfigdb command:

```
sysconfigdb -m -f /mypath/vb_sysconfigtab  vb
```

10. Reboot the vb box manager node. During the boot, the vb driver becomes available and prints VB: messages on the console, including the following message:

```
VB: This is the box manager node
```

When you configure Node 1, do not modify any VMEbus A24 or A16 window attributes in /etc/sysconfigtab, except for the A24 client communication queues offset. For A24_Base, A24_Size, and A16_Base, use the defaults, which do not overlap with the values reconfigured for the box manager node. This will produce the following setup for Node 0 and Node 1:

	A24 Base	Client Queues A24 Address	A24 End	A16 Base
Node 0:	0xA00000	0xBC0000	0xBFFFFF (2 MB)	0x000
Node 1:	0xC00000	0xFC0000	0xFFFFFF (4 MB)	0x100

To configure Node 1, perform the following steps:

1. Create a vb_sysconfigtab file fragment corresponding to Node 0's for Node 1, changing the VB startup state (vb parameter VB_Startup_State) from 0 (off) to 1 (on).

2. Specify the VB node's Ethernet hardware address (vb parameter VB_Netid). For example, if the node's Ethernet address is 08-00-26-E2-24-50, you would specify that address as an unquoted ASCII string.

3. Modify the client communication queues offset, VB_Client_Vme_Window_Offset, to map client queues at the top of the A24 window, based

on Node 1's A24 window size. Specify the value 0x3C0000, which equals the default A24 window size (0x400000) minus the 256 KB needed for client queues (0x040000).

4. The `/mypath/vb_sysconfigtab` file fragment for Node 1 now contains the following text:

```
vb:
   VB_Startup_State = 1
   VB_Netid = 08-00-26-e2-24-50
   VB_Client_Vme_Window_Offset = 0x3C0000
```

   Close the file, then merge its contents into `/etc/sysconfigtab` by issuing the following `sysconfigdb` command:

```
sysconfigdb -m -f /mypath/vb_sysconfigtab  vb
```

5. Reboot Node 1. You should see `VB:` messages printed on the console, including the following message:

```
VB: Box mgr address space is not configured for this system,
    thus this node is not the box manager node (OK).  Be sure
    that there is a box manager in the network.
```

Note

Because Node 1 is using the system defaults for the VMEbus A24 window, you must make sure that if you bring up an additional node (Node 2), you modify the addresses such that the defaults are not used. Even if Node 2 does not turn on the backplane driver, its inbound window overlaps with Node 1. Accesses to the window could cause a system crash or could cause error messages to be printed to the screen of Node 2, because Node 2 is receiving inbound VMEbus accesses from other nodes on addresses to which it has not mapped inbound.

In summary, you should always reconfigure the VMEbus addresses to be unique, no matter how you plan to use the VMEbus.

3.7 UNIVERSE II Two-Node Network Example

The following steps show an easy way to configure two UNIVERSE II nodes to run in a VMEbus backplane (`vb`) network: Node 0, which is the box manager and the system controller, and Node 1.

For each node, most of the UNIVERSE II and `vb` defaults listed in Table 2–4 and Table 3–1 are retained, including most inbound and outbound window characteristics. You should examine the attribute defaults listed in Table 2–4 and Table 3–1, invoke `sysconfigdb -l vba_univ` and `sysconfigdb -l vb` on each node to uncover any previous changes to those defaults, and decide which attribute values require further modification.

In this example, UNIVERSE II and vb parameters that must be modified include the following:

- Inbound and outbound window base addresses
- Mailbox-interrupt window attributes
- The startup state
- The node's Ethernet hardware address
- The client queues offset from the client communication window base address
- The box manager node's well-known VMEbus address

Configure the box manager node first. This example assumes that the box manager node is the VMEbus system controller node.

To configure Node 0, perform the following steps:

1. On Node 0, create a sysconfigtab file fragment in a private directory; for example, /mypath/univ_sysconfigtab. Insert the label vba_univ: at the beginning of the file. In the next few steps, you will construct an indented list, immediately following the label, of the vba_univ attributes you wish to modify and their values.

   This example assumes /etc/sysconfigtab contains no previous vba_univ entry. If such an entry exists, you can either remove the old entry (sysconfigdb -d) before adding the new, or merge the new attributes in with the old (sysconfigdb -m -f). You may need to factor the earlier vba_univ attribute values into your new modifications.

2. Verify that inbound VME-to-PCI (VMEbus slave) windows 0 and 1 are configured at their default VMEbus base addresses, 0x00C00000 and 0x08000000. Node 1's corresponding windows will be relocated to different VMEbus addresses (0x00800000 and 0x10000000). For both Node 0 and Node 1, all other attributes of these windows are left at their defaults.

   In step 12, you will modify the box manager's well-known VMEbus address to map box manager data at the top of VME-to-PCI window 0.

3. Relocate outbound PCI-to-VME (PCI slave) windows 0 through 3 to VMEbus base address 0x10000000, leaving all other window attributes at their defaults. You do this by entering the value 0x10000000 for the vba_univ parameters VME_Wnd0_VME_Address, VME_Wnd1_VME_Address, VME_Wnd2_VME_Address, and VME_Wnd3_VME_Address.

4. Configure the outbound PCI-to-VME windows 4 and 5 to encompass all of A24 address space for user-data and supervisory-data accesses to

other nodes in the system. Because the windows are set up by default for user data and supervisory data, respectively, you only need to specify a new base address, 0x00000000, and a new size, 16 MB, for each. Set VME_Wnd4_VME_Address and VME_Wnd5_VME_Address to the value 0x00000000, and set VME_Wnd4_Size and VME_Wnd5_Size to the value 0x01000000. In addition to providing a complete view of A24 address space for supervisory-data and user-data access, this mapping allows use of MBLTs and data widths up to D64.

5. Configure the outbound PCI-to-VME window 6 as a mailbox-interrupt window. To do this, you design a region of VMEbus space that encompasses the UNIVERSE II adapter CSRs (4 KB per node), including mailbox-interrupt CSRs, for both Node 0 and the partner node, Node 1.

   In this case, the base address of the mailbox-interrupt window will be 0xFFFF0000, its size will be 64 KB, and Node 0 and Node 1 will map their adapter CSRs at 0xFFFF0000 and 0xFFFF1000, respectively. (If location monitors were in use, you could place adapter CSRs at 0xFFFF0000 and 0xFFFF2000, and location monitors at 0xFFFF1000.)

   Set VME_Wnd6_Ena to the value 1, set VME_Wnd6_VME_Address to the value 0xFFFF0000, and set the window size (parameter VME_Wnd6_Size) to 64 KB (value 0x00010000).

   Additionally, you must modify the mailbox-interrupt window's attributes to be compatible with the address modifier attributes of the node's CSR window, which will be mapped in the next step. Set the VME_Wnd6_AM_Space parameter to specify A32 space (value 2) and set the VME_Wnd6_AM_Usr_Sprvsr parameter to specify supervisory mode (value 2). Data access remains selected by default.

6. Configure the node's CSR window in accordance with the design of the mailbox-interrupt window configured in the previous step. For node 0, retain all CSR window defaults, including the VMEbus base address of 0xFFFF0000, A32 space, supervisory mode, and both program and data access.

7. The /mypath/univ_sysconfigtab file fragment now contains the following text:

```
vba_univ:
        PCI_Wnd0_VME_Address = 0x00C00000
        PCI_Wnd1_VME_Address = 0x08000000
        VME_Wnd0_VME_Address = 0x10000000
        VME_Wnd1_VME_Address = 0x10000000
        VME_Wnd2_VME_Address = 0x10000000
        VME_Wnd3_VME_Address = 0x10000000
        VME_Wnd4_VME_Address = 0x00000000
        VME_Wnd4_Size = 0x01000000
        VME_Wnd5_VME_Address = 0x00000000
        VME_Wnd5_Size = 0x01000000
        VME_Wnd6_Ena = 1
        VME_Wnd6_VME_Address = 0xFFFF0000
        VME_Wnd6_Size = 0x00010000
```

```
        VME_Wnd6_AM_Space = 2
        VME_Wnd6_AM_Usr_Sprvsr = 2
```

Close the file, then add its contents to `/etc/sysconfigtab` by issuing the following command:

```
sysconfigdb -a -f /mypath/univ_sysconfigtab  vba_univ
```

8. Create a `sysconfigtab` file fragment for `vb` attributes; for example, `/mypath/vb_sysconfigtab`. (If you want to use the default `vb:` entry provided by `/etc/sysconfigtab` as a starting point, you can copy the existing entry into the file fragment using the command `sysconfigdb -l vb > /mypath/vb_sysconfigtab`.)

   In the next few steps, you will construct an indented list, immediately following the label `vb:`, of the `vb` attributes you wish to modify and their values.

9. Change the VB startup state (`vb` parameter `VB_Startup_State`) from 0 (off) to 1 (on).

10. Specify the VB node's Ethernet hardware address (`vb` parameter `VB_Netid`). For example, if the node's Ethernet address is 08-00-26-E2-48-47, you would specify that address as an unquoted ASCII string.

11. Modify the client communication queues offset, `VB_Client_Vme_Window_Offset`, to map client queues at the top of a 4 MB window. Mapping at the top of a window reduces window fragmentation and minimizes potential conflicts with the memory needs of other VMEbus drivers. Specify the value 0x003C0000, which equals the window size (0x00400000) minus the 256 KB needed for client queues (0x00040000).

12. Modify the box manager well-known address, `VB_Box_Mgr_WK_Addr`, to map box manager data at the top of VME-to-PCI window 0. In step 2, VME-to-PCI window 0 was configured to encompass the box manager well-known address that is shared among all nodes. Specify the value 0x00FC0000, which equals the VME-to-PCI window 0 base address (0x00C00000), plus its size (0x00400000), minus the 256 KB needed for the box manager's VMEbus global data (0x00040000).

13. Mailboxes reside within the CSR window. You must modify the `vb` mailbox-interrupt address type parameter, `VB_Mailbox_Addr_Type`, to match the address modifier attributes associated with the CSR window you configured. Specify A32 address space, supervisory mode, and data access (value 0x6).

14. The `/mypath/vb_sysconfigtab` file fragment now contains the following text:

```
vb:
   VB_Startup_State = 1
   VB_Netid = 08-00-26-e2-48-47
```

```
VB_Client_Vme_Window_Offset = 0x003C0000
VB_Box_Mgr_WK_Addr = 0x00FC0000
VB_Mailbox_Addr_Type = 0x6
```

Close the file, then merge its contents into `/etc/sysconfigtab` by issuing the following `sysconfigdb` command:

```
sysconfigdb -m -f /mypath/vb_sysconfigtab  vb
```

15. Reboot the `vb` box manager node. During the boot, the `vb` driver becomes available and prints `VB:` messages on the console, including the following message:

```
VB: This is the box manager node
```

When you configure Node 1, you should specify UNIVERSE II and `vb` parameter values that you have carefully selected to fit well with the values specified for Node 0, and try to anticipate the needs of any additional nodes that might be added to the `vb` network later. For example, the values specified in this example produce the following setup for Node 0 and Node 1:

Parameter	Node 0	Node 1
VME-to-PCI (inbound) window 0 address	0x00C00000	0x00800000
VME-to-PCI (inbound) window 1 address	0x08000000	0x10000000
PCI-to-VME (outbound) window 0 address	0x10000000	0x08000000
PCI-to-VME (outbound) window 1 address	0x10000000	0x08000000
PCI-to-VME (outbound) window 2 address	0x10000000	0x08000000
PCI-to-VME (outbound) window 3 address	0x10000000	0x08000000
PCI-to-VME (outbound) window 4 address	0x00000000	0x00000000
PCI-to-VME window 4 size	0x01000000	0x01000000
PCI-to-VME (outbound) window 5 address	0x00000000	0x00000000
PCI-to-VME window 5 size	0x01000000	0x01000000
PCI-to-VME (mailbox-interrupt) window 6 address	0xFFFF0000	0xFFFF0000
PCI-to-VME window 6 size	0x00010000	0x00010000
PCI-to-VME window 6 address modifiers	2 (A32), 2 (supervisory), 1 (data)	2 (A32), 2 (supervisory), 1 (data)
CSR window address	0xFFFF0000	0xFFFF1000
Box manager well-known address	0x00FC0000	0x00FC0000
Client queues offset (assumes a 4 MB window)	0x003C0000	0x003C0000

To configure Node 1, perform the following steps:

1. On Node 1, create a `sysconfigtab` file fragment corresponding to Node 0's in a private directory; for example, `/mypath/univ_sysconfigtab`. Insert the label `vba_univ:` at the beginning of the file. In the next few steps, you will construct an indented list, immediately following the label, of the `vba_univ` attributes you wish to modify and their values.
2. Relocate inbound VME-to-PCI (VMEbus slave) windows 0 and 1 to VMEbus locations that differ from those used by Node 0's corresponding windows, leaving all other window attributes at their defaults. Node 0 used the default VMEbus base addresses 0x00C00000 and 0x08000000 for its VME-to-PCI windows 0 and 1. For Node 1, enter the values 0x00800000 and 0x10000000 for the `vba_univ` parameters `PCI_Wnd0_VME_Address` and `PCI_Wnd1_VME_Address`.
3. Relocate outbound PCI-to-VME (PCI slave) windows 0 through 3 to VMEbus base address 0x08000000, leaving all other window attributes at their defaults. Enter the value 0x08000000 for the parameters `VME_Wnd0_VME_Address`, `VME_Wnd1_VME_Address`, `VME_Wnd2_VME_Address`, and `VME_Wnd3_VME_Address`.
4. As with Node 0, configure Node 1's outbound PCI-to-VME windows 4 and 5 to encompass all of A24 address space for user-data and supervisory-data accesses to other nodes in the system. Because the windows are set up by default for user data and supervisory data, respectively, you only need to specify a new base address, 0x00000000, and a new size, 16 MB, for each. Set `VME_Wnd4_VME_Address` and `VME_Wnd5_VME_Address` to the value 0x00000000, and set `VME_Wnd4_Size` and `VME_Wnd5_Size` to the value 0x01000000. In addition to providing a complete view of A24 address space for supervisory-data and user-data access, this mapping allows use of MBLTs and data widths up to D64.
5. Configure the outbound PCI-to-VME window 6 as a mailbox-interrupt window, adhering to the design established during Node 0 configuration. As with Node 0, the base address of the mailbox-interrupt window will be 0xFFFF0000 and its size 64 KB. Node 0 and Node 1 will map their adapter CSRs at 0xFFFF0000 and 0xFFFF1000, respectively.

   Set `VME_Wnd6_Ena` to the value 1, set `VME_Wnd6_VME_Address` to the value 0xFFFF0000, and set the window size (parameter `VME_Wnd6_Size`) to 64 KB (value 0x00010000). As with Node 0, you must modify the mailbox-interrupt window's attributes to be compatible with the address modifier attributes of the node's CSR window, which will be mapped in the next step. Set the `VME_Wnd6_AM_Space` parameter to specify A32 space (value 2) and set the `VME_Wnd6_AM_Usr_Sprvsr` parameter to specify supervisory mode (value 2). Data access remains selected by default.

6. Configure the node's CSR window in accordance with the design of the mailbox-interrupt window configured in the previous step. In this case, only the VMEbus base address needs modification; set CSR_VME_Address to the value 0xFFFF1000. Retain defaults for all other CSR window attributes, including A32 space, supervisory mode, and both program and data access.

7. The /mypath/univ_sysconfigtab file fragment now contains the following text:

```
vba_univ:
        PCI_Wnd0_VME_Address = 0x00800000
        PCI_Wnd1_VME_Address = 0x10000000
        VME_Wnd0_VME_Address = 0x08000000
        VME_Wnd1_VME_Address = 0x08000000
        VME_Wnd2_VME_Address = 0x08000000
        VME_Wnd3_VME_Address = 0x08000000
        VME_Wnd4_VME_Address = 0x00000000
        VME_Wnd4_Size = 0x01000000
        VME_Wnd5_VME_Address = 0x00000000
        VME_Wnd5_Size = 0x01000000
        VME_Wnd6_Ena = 1
        VME_Wnd6_VME_Address = 0xFFFF0000
        VME_Wnd6_Size = 0x00010000
        VME_Wnd6_AM_Space = 2
        VME_Wnd6_AM_Usr_Sprvsr = 2
        CSR_VME_Address = 0xFFFF1000
```

   Close the file, then add its contents to /etc/sysconfigtab by issuing the following command:

```
sysconfigdb -a -f /mypath/univ_sysconfigtab  vba_univ
```

8. Create a sysconfigtab file fragment corresponding to Node 0's for vb attributes; for example, /mypath/vb_sysconfigtab. (If you want to use the default vb: entry provided by /etc/sysconfigtab as a starting point, you can copy the existing entry into the file fragment using the command sysconfigdb -l vb > /mypath/vb_sysconfigtab.)

   In the next few steps, you will construct an indented list, immediately following the label vb:, of the vb attributes you wish to modify and their values.

9. Change the VB startup state (vb parameter VB_Startup_State) from 0 (off) to 1 (on).

10. Specify the VB node's Ethernet hardware address (vb parameter VB_Netid). For example, if the node's Ethernet address is 08-00-26-E2-24-50, you would specify that address as an unquoted ASCII string.

11. Modify the client communication queues offset, VB_Client_Vme_Window_Offset, to map client queues at the top of a 4 MB window. Mapping at the top of a window reduces window fragmentation and minimizes potential conflicts with the memory needs of other VMEbus

drivers. Specify the value 0x003C0000, which equals the window size (0x00400000) minus the 256 KB needed for client queues (0x00040000).

12. Modify the box manager well-known address, VB_Box_Mgr_WK_Addr, to map box manager data at the top of Node 0's VME-to-PCI window 0. Node 0's VME-to-PCI window 0 was configured to encompass the box manager well-known address that is shared among all nodes. Specify the value 0x00FC0000, which equals Node 0's VME-to-PCI window 0 base address (0x00C00000), plus its size (0x00400000), minus the 256 KB needed for the box manager's VMEbus global data (0x00040000).
13. As with Node 0, you must modify the vb mailbox-interrupt address type parameter, VB_Mailbox_Addr_Type, to match the address modifier attributes associated with the CSR window you configured. Specify A32 address space, supervisory mode, and data access (value 0x6).
14. The /mypath/vb_sysconfigtab file fragment now contains the following text:

```
vb:
   VB_Startup_State = 1
   VB_Netid = 08-00-26-e2-24-50
   VB_Client_Vme_Window_Offset = 0x003C0000
   VB_Box_Mgr_WK_Addr = 0x00FC0000
   VB_Mailbox_Addr_Type = 0x6
```

    Close the file, then merge its contents into /etc/sysconfigtab by issuing the following sysconfigdb command:

```
sysconfigdb -m -f /mypath/vb_sysconfigtab  vb
```

15. Reboot Node 1. You should see VB: messages printed on the console, including the following message:

```
VB: Box mgr address space is not configured for this system,
    thus this node is not the box manager node (OK).  Be sure
    that there is a box manager in the network.
```

3.8 Related ioctl Commands

The host's Internet address is specified at boot time with an SIOCSIFADDR ioctl command. The vb interface employs the address resolution protocol described in arp(7) to map dynamically between Internet and Ethernet addresses on the local network.

Use the SIOCRPHYSADDR ioctl command to read the physical address of the VMEbus backplane node. The SIOCSPHYSADDR command cannot be used to change the physical address of the VMEbus backplane node. The VMEbus backplane network does not support DECnet.

Use the SIOCADDMULTI and SIOCDELMULTI ioctl commands to add or delete multicast addresses. The VMEbus backplane driver recognizes a maximum of 64 multicast addresses.

Use the SIOCRDCTRS and SIOCRDZCTRS `ioctl` commands to read or "read and clear" the Ethernet driver counters. The argument to these two commands is a pointer to a counter structure, `ctrreq`, found in `<net/if.h>`.

Use the SIOCENABLBACK and SIOCDISABLBACK `ioctl` commands to enable and disable the interface loopback mode.

To obtain the physical address of the adapter, use the SIOCRPHYSADDR command as in the following program example:

```
#include <stdio.h>              /* Standard I/O */
#include <errno.h>              /* Error numbers */
#include <sys/socket.h>         /* Socket definitions */
#include <sys/ioctl.h>          /* ioctl commands */
#include <net/if.h>             /* Generic interface structures */

main()
{
  int s,i;
  static   struct  ifdevea  devea;
  /* Get a socket */
  s = socket(AF_INET,SOCK_DGRAM,0);
  if (s < 0) {
     perror("socket");
     exit(1);
  }
  strcpy(devea.ifr_name,"vb0");
  if (ioctl(s,SIOCRPHYSADDR,&devea) < 0) {
     perror(&devea.ifr_name[0]);
     exit(1);
  }
  printf("Address is ");
  for (i = 0; i < 6; i++)
     printf("%X ", devea.default_pa[i] & 0xff);
  printf("\\n");
  close(s);
}
```

3.9 Diagnostic Messages

The following diagnostic messages contain relevant information provided by the VMEbus backplane driver, and are not errors:

```
VB: VME Backplane Driver
The backplane driver is not configured to run.
Reconfigure the VB_Startup_State attribute to 1
in the vb: backplane driver subsystem in sysconfigtab.
Driver exiting....
```

The VMEbus backplane driver is not configured on this system. This is the initial default state of the VMEbus driver, before you configure it to run by setting `VB_Startup_State` to 1.

```
VB: VME Backplane Driver
Mailbox interrupts are configured to use A24 space
AND A16 space, which is illegal.
Defaulting to A16 SUPER DATA space for mailbox interrupts.
```

The vb driver attributes specified in /etc/sysconfigtab use an illegal combination of address modifiers for the VMEbus window that maps mailbox interrupts. The driver has reverted to a default set of address modifiers: A16 address space, supervisory mode, and data space.

```
VB: VME Backplane Driver
VB_MAXMTU is outside the allowable range
Setting VB_MAXMTU = 1500
```

The value specified for the vb driver attribute VB_Maxmtu is less than 1500 or greater than 16K and has been reset to the default value, 1500.

```
VB: VB_Maxmtu changed to match the Box manager's MTU n
```

This message is displayed during vb client registration if the vb driver attribute VB_Maxmtu on the client is not equal to VB_Maxmtu on the box manager node. The client value is reset to match the box manager value.

```
VB: This is the box manager node
```

This node's VMEbus address space contains the user-configured address for the box manager node as specified in the sysconfigtab file. Therefore, this is the box manager node. One and only one node in a backplane network should have this message appear at startup.

```
VB: network started
```

This message will appear on a node that has successfully entered the backplane network.

```
VB: shutdown
```

This message will appear when a node in the VMEbus backplane network is shut down. This is a normal diagnostic message.

3.10 Errors

This section lists and describes error messages displayed during and after system startup.

3.10.1 System Startup Error Messages

The following error messages may appear at system startup:

```
VB: Ethernet address contains all zeroes!  DRIVER EXITING...
```

The backplane driver has been configured to be turned on, but the Ethernet address in the file sysconfigtab has not been changed to reflect the Ethernet hardware address of the node. This information must be supplied in order for the node to be entered in the VMEbus backplane network.

```
VB: Incorrect ident in box manager memory.
```

Another device is mapped to the address specified as the box manager well-known address in the sysconfigtab file. Be sure to reconfigure the

box manager address such that it does not overlap another device's CSR address range.

```
VB: VME Backplane Driver
Doorbell interrupts are configured to use A16 space, which
is not the case on this system.
Reconfigure the VB_Mailbox_Addr_Type attribute in sysconfigtab to
use the correct address space according to this system's setup.
Driver exiting....
```

Reconfigure mailbox interrupts as instructed.

3.10.2 Post-Startup Error Messages

The following error messages may appear after system startup:

```
VB: MALLOC failure on box mgr memory
VB: MALLOC failure on l3 queues
```

These messages indicate that the vb driver was unable to allocate memory for internal data structures.

```
VB: Error in dma_get_maps.
```

The vb driver was unable to obtain VMEbus slave window mapping information.

```
VB: Error mapping box mgr memory inbound on the VME.
VB: Error mapping l3 queues inbound on VME.
VB: Error mapping outbound to box mgr
VB: Error mapping outbound to node %d
```

These VMEbus mapping errors are generally caused by misconfigured systems on the backplane network.

```
vb%d: initialization error
```

The vb driver was unable to initialize the network interface.

```
vb%d SIOCADDMULTI fail, multicast list full
```

Too many multicast requests have been made.

Index