VXI/VME-PCI8022 FOR SOLARIS 2.*x*

This document contains information to help you understand the components of your kit, determine where to start setting up your kit, and learn about the NI-VXI/VISA features.

Contents

What Do You Have?

Your VXI/VME-PCI8022 kit contains hardware, software, and documentation. You may also have ordered some optional equipment or software.

Hardware

Your hardware includes the PCI-MXI-2, which you install in your PCI-based computer. You also get either the VXI-MXI-2 or VME-MXI-2 mainframe extender, which you install in your VXI or VME mainframe.

- You do not receive any hardware if you ordered the NI-VXI/VISA software-only kit.

Software

The NI-VXI bus interface software for the PCI-MXI-2 and Solaris 2.*x* includes a Resource Manager, graphical and text-based versions of an interactive VXI resource editor program, a comprehensive library of software routines for VXI/VME programming, and graphical and text-based versions of an interactive control program.

NI-VISA has a comprehensive library of software routines not only for VXI/VME programming, but also for GPIB, GPIB-VXI, and Serial. You can use this software to program instruments connected via different types of interfaces.

Documentation

With the exception of the *Installation Guide* in your kit, all documentation is online. The following list shows the path and filenames of the online manuals that are available after you install your software:

- `/opt/NICpcimxi/manuals/GettingStarted.pdf`
  This contains an overview of the MXI-2 hardware and the NI-VXI software, guides you through setting up your kit and configuring the hardware and software, and helps you get started with application development. You can find a *Quick Start* section in Chapter 1, *Introduction and Quick Start*, that summarizes the setup instructions and lists the hardware and software default settings. Appendix D, *Common Questions*, addresses commonly asked questions you may have about using the NI-VXI software on the PCI-MXI-2 platform.
- `/opt/NICpcimxi/manuals/NI-VXIUsersMan.pdf`
  This is the *NI-VXI User Manual*, which describes the features of the NI-VXI software.
- `/opt/NICpcimxi/manuals/NI-VXIProgrammerMan.pdf`
  This is the *NI-VXI Programmer Reference Manual*, which describes in detail the VXI/VME function calls in the C/C++ and BASIC languages.
- `/opt/NICpcimxi/manuals/GraphicalUtils.pdf`
  This is the *NI-VXI Graphical Utilities Reference Manual*, which

describes the graphical NI-VXI utilities VXIedit, VICVIC, and the Startup Resource Manager.

- `/opt/NICpcimxi/manuals/TextUtil.pdf`
  This is the *NI-VXI Text Utilities Reference Manual*, which describes the text-based NI-VXI utilities VXItedit, VICtext, and the Startup Resource Manager.
- `/opt/vxipnp/sun/NIvisa/Manuals/NI-VISAUsersMan.pdf`
  This is the *NI-VISA User Manual*, which describes how to program using VISA.
- `/opt/vxipnp/sun/NIvisa/Manuals/NI-VISAProgrammersMan.pdf`
  This is the *NI-VISA Programmer Reference Manual*, which describes in detail the attributes, events, and operations you use in NI-VISA.

Available Options

You may have ordered the following optional software or accessories:

- LabVIEW
- LabWindows/CVI
- MXI-2 cable

♦ These options are not available if you ordered the NI-VXI/VISA software-only kit.

Both LabVIEW and LabWindows/CVI integrate the VXI and VISA library interfaces that are required to support your PCI-MXI-2 products. You also get hundreds of complete instrument drivers, which are modular, source-code programs that handle the communication with your instrument to speed your application development.

Where Do You Start?

1. Compare your kit contents with the description in the preceding *What Do You Have?* section. Contact National Instruments regarding any discrepancies.
2. Install LabVIEW or LabWindows/CVI before you install NI-VXI/VISA if you intend to use either of these application development environments as your programming choice. The NI-VXI/VISA installer will update necessary LabVIEW and LabWindows/CVI files required to interface to your VXI/VME system.
3. Install your MXI-2 hardware into your computer and mainframe. Refer to Chapters 2 through 4 in *Getting Started with Your VXI/VME-PCI8020 and the NI-VXI Software for Solaris* for full

instructions on configuring and installing the PCI-MXI-2, VXI-MXI-2, and VME-MXI-2, respectively.

4. Install the NI-VXI/VISA software as described in Chapter 5, *NI-VXI Software Installation*, in your getting started manual. Although the manual does not mention NI-VISA, the installation procedure can install both NI-VXI and NI-VISA.
5. Install the HyperHelp Runtime Environment. This file is already installed for you if you have LabVIEW 4.*x* or higher installed on your system. You must install this file to access the NI-VISA online help. Disregard this step if you are not going to use VISA. Refer to the *Installing the HyperHelp Runtime Environment* section later in this document for installation instructions.
6. The default configuration settings are suitable for the most typical applications. Use VXIedit or VXItedit as described in Chapter 6, *NI-VXI Configuration Utility*, if you want to reconfigure any of the MXI-2 hardware. Use the VISAconf utility to configure any settings specific to VISA.
7. After you finish installing the software and hardware, refer to Chapter 7, *Using the NI-VXI Software*, in your getting started manual to learn how you can use your VXI/VME system and to ensure it is operating properly.
8. Please refer to the following files for important information that may affect your application program, including known issues and software corrections with this release, and additional information relevant for NI-VXI and NI-VISA API development:
   - `/opt/NICpcimxi/README` for NI-VXI information
   - `/opt/vxipnp/sun/NIvisa/README` for NI-VISA information

   You can also reference the National Instruments `ni.com` or `ftp.ni.com` sites for driver updates, examples, and product news.

What Is NI-VXI?

The NI-VXI system-level software is the driver that controls your PCI-MXI-2 interface and VXI/VME system. NI-VXI includes a Resource Manager, libraries of software routines for test and measurement programming and interactive control programs for both NI-VXI and NI-VISA. You can use this software to seamlessly program multiple-mainframe configurations and have software compatibility across a variety of controller platforms.

What Is VISA?

VISA is a standard I/O Application Programming Interface (API) for instrumentation programming. VISA by itself does not provide instrumentation programming functionality for interfaces other than Serial. VISA is a high-level API that calls into system-level drivers. As an example, the NI-VISA implementation of VISA uses the NI-VXI system-level driver for National Instruments VXI controllers. If you are a GPIB or GPIB-VXI user, you must also install the NI-488.2 system-level driver for National Instruments GPIB controllers.

VISA can control VXI, GPIB, or Serial instruments, making the appropriate driver calls depending on the type of instrument being used. VISA uses the same operations to communicate with instruments regardless of the interface type. For example, the VISA command to write an ASCII string to a message-based instrument is the same whether the instrument is Serial, GPIB, or VXI. As a result, VISA gives you interface independence. This makes it easier to switch bus interfaces and means that users who must program instruments for multiple interfaces need learn only one API.

Another advantage of VISA is that it is an object-oriented API that will easily adapt to new instrumentation interfaces as they evolve, making application migration to the new interfaces easy.

Because VISA is the industry standard for developing instrument drivers, most instrument drivers currently written by National Instruments use VISA and therefore support Macintosh, Windows 3.*x*, Windows 9*x*, Windows NT/2000, Solaris 1, Solaris 2, and HP-UX, if the system-level drivers are available for that platform.

NI-VXI/VISA Release Notes

This section describes the new features, enhancements, and supported ADEs in this release of NI-VXI/VISA for Solaris 2.*x*.

Upgrading from an SB-MXI System

Software compiled with the BINARY_COMPATIBLE flag using NI-VXI 2.2 for the SB-MXI is binary compatible with NI-VXI for the PCI-MXI-2. There are some instances, however, that can prevent your application from working with the PCI-MXI-2.

The SB-MXI had fixed window sizes for mapping pointers to A16, A24, and A32 space with `MapVXIAddress()`. The PCI-MXI-2 does not have fixed window sizes. (Refer to the *New Features and Terminology* section

for details about MITE window mapping.) If your application depends upon the mapped window size and does not check the size of a returned window with `GetWindowRange()`, it may not work properly. Always use `GetWindowRange()` after a call to `MapVXIAddress()` to check the size of a window.

Code used with the SB-MXI was compiled with the VXIUNIX flag defined. The compilation flag is now VXISOLARIS for the PCI-MXI-2 driver. If you need to recompile your application, you need to define this new flag.

The PCI-MXI-2 and the NI-VXI software support a memory sharing feature. When you use this feature, the memory on your workstation is visible to other VXI bus masters. The SB-MXI did not support this feature. Consult the getting started manual for help in using slave memory.

The NI-VXI software for the PCI-MXI-2 contains two new utilities that were not provided with the NI-VXI software for the SB-MXI: VIC and VXIedit. These are graphical utilities you can use to interactively control the VXI bus and to configure your MXI-2 hardware. Refer to your getting started manual and to the *NI-VXI Graphical Utilities Reference Manual* for full details on these utilities. Use the Acrobat Reader 3.0 to view and navigate through the *NI-VXI Graphical Utilities Reference Manual*.

Installing the HyperHelp Runtime Environment

Install the hardware and software as described in *Getting Started with Your VXI/VME-PCI8020 and the NI-VXI Software for Solaris*. To access the NI-VISA online help, you must complete your software installation by installing the HyperHelp Runtime Environment. If you have installed LabVIEW 4.*x* or higher on your system, the HyperHelp Runtime Environment is already installed for you; otherwise, follow these instructions:

1. Identify and switch to the directory in which you want to install HyperHelp. For example, if you want to install HyperHelp under the `/opt` directory, enter the following command:

   ```
   cd /opt
   ```

Note For Solaris 2.x, the recommended directory is `/opt`.

2. Type the following command to insert the HyperHelp media into your installation device and copy the HyperHelp archive files:

   ```
   tar xvf /dev/rfd0
   ```

Note The device name shown in this code example may be different for your system. Refer to your system's user guide or system administrator for the correct device name.

Repeat this procedure for all HyperHelp runtime environment and configuration diskettes.

3. Type the following commands to uncompress and unzip the HyperHelp files:

   ```
   uncompress * z
   ./gunzip.sol2 *.gz
   ```

   The `gunzip` utility is distributed under the terms of the GNU General Public License solely for the purpose of uncompressing the HyperHelp files. In accordance with the terms of the license, to obtain source code for `gunzip`, send e-mail to `support@bristol.com`.

4. Type the following command to unarchive each file that ends with the `.tar` extension:

   `tar xvf` *`filename`*`.tar`

5. Set the HHHOME environment variable to the location of the HyperHelp files. Add the following corresponding line to your login file:

   - C Shell Users:

     `setenv HHHOME` *`install_dir`*`/hyperhelp`

   - Korn Shell or Bourne Shell Users:

     `HHHOME=`*`install_dir`*`/hyperhelp;export HHHome`

6. Add `$HHHOME/bin` to your PATH environment variable.

Supported Application Development Environments

This release of NI-VXI/VISA for Solaris 2.*x* supports the following Application Development Environments (ADEs):

- LabVIEW version 4.*x* or higher
- LabWindows/CVI version 4.*x* or higher
- gcc version 2.6.3 or higher
- cc version SC 3.0.1 or higher

Note Although NI-VXI and NI-VISA have been tested and found to work with these ADEs, other ADEs listed above may also work.

New Features and Terminology

New functionality has been added to NI-VXI/VISA in four major areas to exploit features in the MITE ASIC. These features are as follows:

- Window mapping
- MITE DMA

- Shared memory
- Remote controllers

Window Mapping

The MITE architecture allows much more flexibility in low-level mapping of VXI address spaces. In particular, the CPU interface of the MITE has windows that can be dynamically resized and relocated from CPU space to VXI space. The low-level functions have new extensions that reflect this feature. Refer to the NI-VXI online help or the *NI-VXI Programmer Reference Manual* for information about the low-level NI-VXI functions. The NI-VISA online help and the *NI-VISA Programmer Reference Manual* discuss this information for NI-VISA applications. As mentioned earlier in this document, use the Acrobat Reader 3.0 to view and navigate through these manuals.

The functions `MapVXIAddress()` and `viMapAddress()` check whether a sharable window already maps to the desired address space and location. If so, they return a pointer to that window. If the desired space is not already mapped, they set up a new MITE window to the VXI address and return a pointer to the new window.

The `MapVXIAddressSize()` function is the standard mechanism for specifying how large a window the driver should map on a call to `MapVXIAddress()`. The default size of a mapped window when using NI-VXI is 64 KB. In VISA, you specify the window size directly in `viMapAddress()`.

The success of this allocation depends on the availability of three factors:

- Address space in the User Window
- Number of MITE windows
- Memory for allocating data structures for the map

Address Space

The PCI-MXI-2 can decode any 32-bit address on the PCI bus as a VXI cycle, giving 4 GB of addressability, which can be used for windows on the PCI-MXI-2. The operating system or computer architecture may limit which addresses can be assigned to the PCI-MXI-2.

To change the address space, edit the **User Window Size** field in the **Bus Configuration Editor** of the **PCI-MXI-2 Configuration Editor** in VXIedit. This setting limits the total amount of memory you can map with `MapVXIAddress()` or `viMapAddress()`. If the User Window is disabled, the `MapVXIAddress()` function returns `NO_HARDWARE_SUPPORT (-1)`.

The *NI-VXI Programmer Reference Manual* implies that the error code `MAP_TIMEOUT (-8)` is returned when the window is in use. Because the MITE-based products have multiple hardware windows of variable size, the meaning of this error has been modified. `MapVXIAddress()` now returns the error code `MAP_TIMEOUT (-8)` whenever there are not enough resources to map the window.

For example, if you use `MapVXIAddressSize()` and `MapVXIAddress()` to request a 1 MB window to A32 space, and you request a user window in VXIedit of only 64 KB, `MapVXIAddress()` returns the error code `MAP_TIMEOUT` because there are not enough resources to complete the request.

Number of MITE Windows

The MITE has eight CPU windows. NI-VXI uses four of these windows, leaving four for user applications.

Memory for Allocating Data Structures

You need to have sufficient memory available to set up the necessary page tables. If you request a very large window—hundreds of megabytes, for example—you may run out of memory.

MITE DMA

The MITE has two DMA channels to improve the throughput of block transfers to and from the VXI system. The DMA channels can use various high-speed bus protocols, such as the following:

- MXI block
- MXI synchronous
- Burst mode (on the PCI bus)
- VME64 (on the VXI bus)

The DMA channels can transfer data between a VXI device and local memory, or between VXI devices. The DMA channel can handle contiguous or noncontiguous local memory. If it is handling noncontiguous memory, it can perform scatter-gather operations on the noncontiguous memory.

The `VXImove()` and `viMove`*`XX`*`()` functions automatically use appropriate bus protocols and transfer types to efficiently perform the data transfer specified in the function. You can also use the **Bus Configuration Editor** options in VXIedit to instruct the NI-VXI/VISA software to use DMA channels for particular types of operations and to designate what protocols the channel should use. In addition, you can programmatically control which protocols to use in NI-VXI. See the NI-VXI online help or the

NI-VXI Programmer Reference Manual for complete descriptions of `VXImove()` and other high-level functions. Notice that previously written NI-VXI and NI-VISA code uses the DMA capabilities of the MITE without modification.

To take full advantage of the throughput of the DMA channels, you should perform 32-bit transfers where both the source and the destination are longword aligned. If you need to transfer character data between devices of different byte orders—for example, between a big-endian device and an Intel 80*x*86-based Windows NT PC—transfer the data as longwords but adjust the byte-ordering parameters in `VXImove()` to get the correct data in the most efficient manner.

NI-VXI Examples:

```
/* Transferring 32-bit data to a big-endian A32 device */
VXImove(0x0, userBuffer, 0x3, deviceOffset, numDataPoints, 4);

/* Transferring 8-bit data to a big-endian A32 device */
VXImove(0x80, userBuffer, 0x3, deviceOffset, numDataPoints / 4, 4);
```

Shared Memory

In the **Logical Address Editor** settings of the **PCI-MXI-2 Configuration Editor** in VXIedit, you can share memory on your computer or from DRAM added to the PCI-MXI-2. You can access shared memory on your computer using `VXImemAlloc()` in NI-VXI and `viMemAlloc()` in VISA.

Remote Controllers

Remote controllers, when configured to detect asynchronous events such as a VXI interrupt or VXI trigger, need to inform the local controller that such an asynchronous event has occurred. The remote controllers report these events back to the local controller via a VXI IRQ line. This IRQ line is called the *system IRQ line*. You can use VXIedit to select which VXI interrupt line the remote controller uses to report remote events to the local controller. You need to map the system IRQ line back to the local controller to receive remote controller interrupts. This mapping is performed automatically by the Resource Manager in the parent-side VXI-MXI-2 controllers, but not in other mainframe extenders. You can map interrupts through VXIedit, or with the `MapVXIint()` function, which is described in the NI-VXI online help or the *NI-VXI Programmer Reference Manual*.

The system IRQ line is treated differently than other IRQ lines used by NI-VXI:

- The system IRQ line is always acknowledged by the Resource Manager (Logical Address 0).
- The system IRQ line cannot be disabled on the Resource Manager. Calling `DisableVXIint()` on the system IRQ line does *not* disable it.
- Devices other than remote controllers can also interrupt on the system IRQ line, provided that the device at Logical Address 0 is the handler for the interrupt.
- Routing the system IRQ line to the signal queue is not recommended. Because the system IRQ line cannot be disabled, this routing could lose interrupts.

Passing the value –1 as the logical address of a controller in NI-VXI causes NI-VXI to select the first *remote* controller in your system. Notice that on embedded controllers such as the VXIpc-870, –1 refers to the *local* controller. This is to maintain compatibility with older systems where the external controller needed an extender to assert and receive interrupts.

Enhancements to the NI-VXI Software

The following sections describe the additional options beyond what is documented in the *NI-VXI User Manual* and the *NI-VXI Programmer Reference Manual*.

Compatibility

NI-VXI applications that follow the guidelines documented in the *NI-VXI User Manual* will work with NI-VXI for the PCI-MXI-2.

System Configuration Functions

The `InitVXIlibrary()` function has a new return value of `INIT_RET_OK_RMERROR (2)`. If this value is returned, it means “The NI-VXI library successfully initialized, but the Resource Manager has not been run successfully.” Always run the Resource Manager before using the NI-VXI library.

Low-Level VXIbus Access Functions

Do not make any assumptions about the size and features of a window returned from `MapVXIAddress()`. You should use `GetWindowRange()` to determine the size of a window.

The 32-bit value returned from `GetContext()` and passed to `SetContext()` has a new format. Applications that set the context bits directly for use in `SetContext()` may not be compatible with the new format for context. Because the MITE allows more flexible window mapping, extra bits have been added to this field to reflect these new features. Do not manipulate the context bits directly.

High-Level VXIbus Access Functions

For best performance, keep the following in mind when using `VXImove()`:

- Make sure your buffers are 32-bit aligned.
- Transfer 32-bit data whenever possible.
- Using VXI block access privileges significantly improves performance to devices that are capable of accepting block transfers.
- `VXImove()` must lock the user buffer in memory on virtual memory systems, so locking the buffer yourself optimizes `VXImove()`.
- Because `VXImove()` must build a scatter-gather list for the user buffer on paged memory systems, using a contiguous buffer optimizes `VXImove()`.

`VXImemAlloc()` returns 32-bit aligned, page-locked, contiguous buffers, which work efficiently with `VXImove()`, but only if the function returns `MEM_OK (0)`. A status of `MEM_OK_USE_MEMCOPY (1)` means this buffer cannot be used directly with `VXImove()`.

`VXImove()` can also move blocks of data to and from a single VXI address. This is commonly referred to as *FIFO mode*. For more information refer to the *NI-VXI Programmer Reference Manual*.

Local Resource Access Functions

`VXImemAlloc()` does not allocate onboard RAM on the PCI-MXI-2; it only allocates system RAM on the motherboard. If you want to access onboard RAM on the PCI-MXI-2, access it as if it were VXI memory—that is, by using high-level or low-level VXIbus access functions. You can use `GetDevInfo()` on the PCI-MXI-2 device to determine the VXI address space and VXI address of this onboard RAM.