

The High-Performance Alternative

DNA-PPCx PowerDNA Cube User Manual

**Architecture & Configuration of the
Core Module for the DNA-PPCx PowerDNA Cube**

**February 2009 Edition
PN Man-DNA-Core 0209
Version 3.4**

© Copyright 1998-2009 United Electronic Industries, Inc. All rights reserved.

No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form by any means, electronic, mechanical, by photocopying, recording, or otherwise without prior written permission.

Information furnished in this manual is believed to be accurate and reliable. However, no responsibility is assumed for its use, or for any infringements of patents or other rights of third parties that may result from its use.

All product names listed are trademarks or trade names of their respective companies.

See UEI's website for complete terms and conditions of sale:

<http://www.ueidaq.com/company/terms.aspx>

Contacting United Electronic Industries

Mailing Address:

27 Renmar Avenue
Walpole, MA 02081
U.S.A.

For a list of our distributors and partners in the US and around the world, please see

<http://www.ueidaq.com/partners/>

Support:

Telephone: (508) 921-4600

Fax: (508) 668-2350

Also see the FAQs and online "Live Help" feature on our web site.

Internet Support:

Support support@ueidaq.com

Web-Site www.ueidaq.com

FTP Site <ftp://ftp.ueidaq.com>

Product Disclaimer:

WARNING!

DO NOT USE PRODUCTS SOLD BY UNITED ELECTRONIC INDUSTRIES, INC. AS CRITICAL COMPONENTS IN LIFE SUPPORT DEVICES OR SYSTEMS.

Products sold by United Electronic Industries, Inc. are not authorized for use as critical components in life support devices or systems. A critical component is any component of a life support device or system whose failure to perform can be reasonably expected to cause the failure of the life support device or system, or to affect its safety or effectiveness. Any attempt to purchase any United Electronic Industries, Inc. product for that purpose is null and void and United Electronic Industries Inc. accepts no liability whatsoever in contract, tort, or otherwise whether or not resulting from our or our employees' negligence or failure to detect an improper purchase.

Table of Contents

Chapter 1 Introduction	1
1.1 PowerDNA Overview	3
1.1.1 What's in the Package	3
2.1 Overview	4
Chapter 2 Installation and Configuration	6
2.1 Initial Installation - Overview	6
2.2 Initial Installation – Start-to-finish Guide	7
2.2.1 Inspect the package	7
2.2.2 Install Software	7
2.3.3 Initial Boot-up	9
2.2.3 IP Addresses on the PowerDNA Cube	10
2.2.4 Improving Network Performance	11
2.2.5 PowerDNA Explorer Quick-Start	15
2.2.6 Updating Firmware	17
2.2.7 Firmware Update Instructions	18
2.3 Mounting and field connections	20
2.4 Wiring	21
2.5 Peripheral Terminal Panel Wiring	23
2.6 Repairing (and upgrading) Your Cube	23
Chapter 3 PowerDNA Explorer	24
3.1 The Main Window	24
3.2 Menu Bar	24
3.2.1 File Menu	24
3.2.2 Network Menu	25
3.2.3 View Menu	27
3.2.4 Help Menu	28
3.2.5 Toolbar	28
3.2.6 Device Tree	28
3.2.7 Settings Panel	29
3.2.8 Digital Input/Output Layer Settings	31
3.3 Analog Output Layer Settings	37
3.4 Analog Input Layer Settings	38
3.5 Counter/Timer Layer Settings	39
Chapter 4 The PowerDNA Core Module	42
4.1 Device Architecture of DNA-CM	43
4.1.1 Device Architecture of DNA-PPC	44
Chapter 5 Programming Layer-specific Functions	45
5.1 Overview	45
5.2 Memory Map	45
5.2.1 Startup sequence (DNA-CM-5/8)	46
5.2.2 Startup Sequence (DNA-PPC-5/8)	47
5.2.3 Interfacing to the CM Module Using a Serial Interface	47
5.2.4 Setting Parameters	50

5.3	How to Update Firmware	52
5.3.1	Clock and Watchdog Access	52
5.4	Common Layer Interface	52
5.4.1	Channel List	52
5.4.2	Configuration Flags	54
5.4.3	EEPROM User Area Access	56
5.5.4	PowerDNA Layer Signaling	58
5.5	Register Map and Description	60
6.1	Register Descriptions	61
6.1.1	FIFO Access	68
6.1.2	Command Mode	69
Chapter 6 Host / IOM Communication		70
6.1	Host / IOM Communication Modes	70
6.1.1	Synchronous vs. Asynchronous	71
6.2	Buffered I/O	71
6.3	Advanced Circular Buffer (ACB)	72
6.3.1	Burst Mode	74
6.4	Message Mode (Msg Protocol)	74
6.4.1	IOM/Host Data Transfer	74
6.4.2	CAN-503 Data Transfer	75
6.4.3	PDNLib Structures	75
6.4.4	Error Recovery	76
6.4.5	Other Messaging Types	76
6.5	Mapped I/O	77
6.5.1	Fixed-Size Data Mapping (DMap)	77
6.7.2	Variable-size Data Mapping (VMap)	79
6.6	Choosing the Right Layers, Operating System, and Mode	81
6.6.1	Attributes of Modes	81
6.6.2	Application Requirements	83
6.6.3	Selecting the Right Mode for Your Application	85
Chapter 7 How DaqBIOS Protocol Works		87
7.1	DaqBIOS paCket Structure	87
7.2	DaqBIOS Protocol Versions	89
7.3	Host and IOM Data Representa-tion	89
7.3.1	Soft and Hard Real-time	89
7.3.2	DaqBIOS & Network Security	90
Chapter 8 DaqBIOS Engine		91
8.1	Basic Architecture	91
8.2	Threads and Function	92
8.3	IOM Data Retrieval and Data Conversion	92
Chapter 9 Real-time Operation with an IOM		93
9.1	Simple I/O	93
9.2	Real-time Data Mapping (RtDmap)	93
9.2.1	Data Replication over the Network	94
9.2.2	RtDmap Functional Description	94

9.2.3	RtDmap Typical Program Structure	96
9.3	Real-time Variable-size Data Mapping (RtVmap)	97
9.3.1	RtVmap Typical Program Structure	102
Appendix A	104
A.1	Configuring a Second Ethernet Card Under Windows XP.	104
A.2	Configuring a Second Ethernet Card Under Windows 2000	107
A.3	Configuring a Second Ethernet Card Under Windows NT	110
A.4	Configuring a Second Ethernet Card Under Windows 95/98/SE/ME	112
Index	117

List of Figures

Chapter 1 Introduction	1
(None)	
Chapter 2 Installation and Configuration	6
2-1 Typical MTTY Startup Screen	9
2-2 PowerDNA Explorer Startup Screen.....	15
2-3 Update Firmware Menu Item	18
2-4 Password Dialog Box	18
2-5 Firmware Update Progress Dialog Box	19
Chapter 3 PowerDNA Explorer	24
3-1 PowerDNA Explorer Main Window.....	24
3-2 Preferences	24
3-3 Address Ranges Dialog Box.....	25
3-4 Edit Address Ranges Dialog Box	25
3-5 After a Network >>Scan Network	26
3-6 Password dialog box for Store Config and Store All Configs.....	27
3-7 Password Dialog Box for Update Firmware.	27
3-8 Example of a Wiring Diagram.....	28
3-9 Example of the Device Tree	28
3-10 Example of IOM Settings Panel for a PowerDNA cube	29
3-11 Example of Device Layer Settings for a Layer.....	30
3-12 Screen from Network >> Read Input Data	31
3-13 Example DIO-405 Layer Inputs	32
3-14 Example DIO-405 Layer Outputs	33
3-15 Example of DIO-403 Layer Inputs	34
3-16 Example of DIO-403 Layer Outputs	34
3-17 Example of DIO-403 Layer Outputs	35
3-18 Example DIO-403 Layer Configuration.....	35
3-19 Example DIO-403 Layer Initialization	36
3-20 Example AO-302 layer	37
3-21 Example AI-201 layer	38
3-22 Example CT-601 layer.....	39
3-23 Example Quadrature controls	39
3-24 Example Bin Counter controls	40
3-25 Example Pulse Width Modulation (PWM) controls	40
3-26 Example Pulse Period controls.....	40
3-27 Example of Started Counter	41
Chapter 4 The PowerDNA Core Module	42
4-1 PowerDNA Core Module (CPU and NIC)	42
4-2 FreeScale ColdFire Controller Architecture	43
4-3 PowerPC Controller Architecture.....	44
Chapter 5 Programming Layer-specific Functions	45
5-1 Changing the IP Address.....	51
5-2 CM Interconnection Diagram.....	58
Chapter 6 Host / IOM Communication	70
6-1 Communicating with an IOM	70
6-2 Host / IOM Communication in ACB Mode (with DQE).....	72
6-3 Data Field of a RDFIFO Packet Containing Messages	75
6-4 Message Block for CAN messages in FIFO	75
6-5 Host / IOM CCommunication in DMap Mode	77
6-6 Host / IOM Communication in VMap Mode (with DQE).....	79

Chapter 7 How DaqBIOS Protocol Works	87
7-1 DaqBIOS Packet Over UDP Packet	87
7-2 DaqBIOS Packet Over Raw Ethernet Packet	87
Chapter 8 DaqBIOS Engine	91
8-1 User Application/DQE/IOM Interaction	91
Chapter 9 Real-time Operation with an IOM	93
9-1 DMap Operation	93

Chapter 1 Introduction

This document is intended to serve as a user manual for a PowerDNA Cube system. It describes the PowerDNA Cube Distributed Network Acquisition system, its components, specifications, and instructions for set up and operation.

PowerDNA is the umbrella name that describes a real-time distributed I/O system with exceptional flexibility and performance. PowerDNA system consists of three parts: (1) Input/Output Modules (a.k.a. I/O Modules, IOMs, Cubes) distributed throughout a process, large piece of equipment, facility, or other structure; (2) Cubes connected via copper -or- fiber optic cables to (3) a host PC with a dedicated Ethernet interface card and running Windows, Linux, or an RTOS. Cubes may also be operated in stand-alone data-logger mode.

The PowerDNA Cube is available in either a 5- or 8-layer configuration. Two of these layers are occupied by the Core Module. The Core Module consists of the CPU Layer and the NIC (network-interface control) Layer, with connectors for either 100Base-T copper or 100Base-FX fiber-optic cable. The remaining 3 or 6 slots in the Cube are factory-configured with your selection of I/O Layers. For information on these data-acquisition layers, visit www.ueidaq.com.

This document gives further details about the features and functions of various system components. Details on programming the system are contained in the companion document(s): the PowerDNA API Reference Manual, and various layer manuals.

Who should read this manual?

This manual has been written to make the installation, configuration, and operation of the PowerDNA cube as straightforward as possible. However, it assumes that the user has basic PC skills and is familiar with the Microsoft Windows XP/2000/ NT/9x, QNX or Linux/RTLinux/RTAI Linux operating environments.

Organization of this manual

This PowerDNA User Manual is organized as follows:

- **Chapter 1—Introduction**
An introduction to the cube.
- **Chapter 2—Installation and Configuration**
Provides instructions for installing and configuring the cube
- **Chapter 3—The PowerDNA Explorer**
Provides an overview of PowerDNA Explorer Main Window, menu bar, toolbar, Device Tree, setting panel, IOM settings, and Device layer settings.
- **Chapter 4—PowerDNA Core Module**
Describes the function and architecture of the CPU and NIC layers

- **Chapter 5—Programming Layer-specific Functions**
Describes device architecture, memory map, startup sequence, setting parameters, updating firmware, common layer interface.
- **Chapter 6—Host / IOM Communication**
Describes various modes of communication between host and IOM and how to select the one best suited for your application.
- **Chapter 7—How DaqBIOS Protocol Works**
Describes how packets of information are transferred over the Ethernet between IOM and host.
- **Chapter 8—The DaqBIOS Engine**
Describes the DaqBIOS Engine, a set of functions and data structures that implement the DaqBIOS data acquisition protocol.
- **Chapter 9—Real-time operation with an IOM**
Describes data mapping and streaming under control of a real-time operating system. It discusses low level programming of such operations without using the DQE.
- **Appendix**
Provides an overview of how to determine the version of PowerDNA, update the firmware, and configure the Ethernet card in various Windows OS and Linux installations.
- **Index**
Alphabetical listing of the topics covered in this manual.

Conventions

To help you get the most out of this manual and our products, please note that we use the following conventions:



Tips are designed to highlight quick methods to get the job done, or to reveal uncommon knowledge and ideas.

NOTE: Notes alert you to important information.



CAUTION! *Caution advises you of precautions to take to avoid injury, data loss, or a system crash.*

Text formatted in **bold** typeface generally represents text that should be entered verbatim. For instance, it can represent a command, as in the following example: “Instruct operator of how to run setup using a command such as **setup.exe**”

Other PowerDNA Documentation

This *PowerDNA User Manual* is one part of the documentation set available for the PowerDNA system. We offer other resources you might want to read before programming an application. They are available either on the PowerDNA Software Suite CD or can be downloaded from the UEI web site.

In particular, we recommend the *PowerDNA API Reference Manual*, *PowerDNA Quick Start Manual*, *UEIDAQ Framework Reference Manual*, *UEIDAQ Framework User Manual*, and the *UEIDAQ Framework Getting Started Manual*.

Feedback

We are interested in any feedback you might have concerning our products and manuals. Comments and recommendation can be sent by email to support@ueidaq.com.





1.1 PowerDNA Overview

This chapter provides an overview of the key features of the PowerDNA system, and how the system works.

Thank you for purchasing a PowerDNA Cube system. We designed this product family from the ground up to provide the best possible features, reliability, and performance at an economically sound price.

1.1.1 What's in the Package

Inspect the package. Included you should find:



	The PowerDNA Cube Preinstalled with your selection of I/O Layers
	Power supply (DNA-PSU-24: 100-240V 50-60Hz to 24VDC)
	Ethernet cable with either RJ-45 connector (for copper) or SC-type (for fiber optic 100-Base-FX cable)
	Serial cable (for initial configuration)

Additional accessories may be included, depending on your order.

1.2 Overview

The PowerDNA system consists of a hardware Cube and software suite. The software suite is located both on the PowerDNA / PowerDAQ CD shipped with the Cube and on the website: www.ueidaq.com

The software that supports the system consists of two components:

	PowerDNA Software Suite	PowerDNA low-level driver; PowerDNA Explorer (and demo); Example code for C & Java
	UEIDAQ Framework	Additional example code & docs for C/C++, C#, VB.NET, ActiveX (VB6, Delphi), MATLAB, LabVIEW, DASyLab, LabWindows/CVI, OPC

The Windows PowerDNA Software Suite contains the following software:

- **PowerDNA low-level driver**
The interface between the cube hardware and higher-level languages.
- **PowerDNA Explorer**
The essential tool for configuring and testing the cube. See Chapter 3 for use.
- **Multi-Threaded TTY Client**
For initial setup of the cube on the network, upgrades, and calibration.
- **Example C & Java code**
Facilitates jumping in and learning — this code will compile and execute on the cube.

In addition to the examples in the PowerDNA Software Suite, the UEIDAQ Framework contains example code for higher-level languages (C++, VB, Java), and also several graphical programming languages (e.g., LabVIEW, DASyLab).

The framework facilitates and expedites test development: an experiment can be set up in less than twenty lines of code. The framework function calls are portable between programming languages.

The Linux software package includes:

- **DAQLib** - Library for writing programs using PowerDNA IO modules (cubes)
- **UeiPalib** - Platform abstraction library needed for building the DAQLib
- **DAQLib_Samples** - Example programs demonstrating how to use the DAQLib to work with various layer types

Instructions on use can be found in the readme.txt of the package.

The hardware / PowerDNA cube is composed of:

- External casing – in two compact sizes:
Core Module + 3 I/O Layers: 3.95" × 4.1" × 4.0"
Core Module + 6 I/O Layers: 5.8" × 4.1" × 4.0"
- Core Module [2 layers at the top]
 - The CPU Layer [PowerPC | Coldfire]
Integrated CPU with real-time kernel in firmware;
Cube can operate as a standalone unit
 - The NIC Layer [100BaseT | Fiber 100-Base-FX]
Can ILink cube to any PC over commercial Ethernet,
Daisychain 64 Cubes over one Ethernet network
- Optional I/O-layers (refer to www.ueidaq.com for details)
- Resolutions to 24 bits; read/write to a Cube's I/O Layers every 1 msec
- Analog Input
 - High-gain & low-gain
 - Strain Gauge module
 - Simultaneous Sampling module
- Analog Output
with optional current/voltage booster add-in card
- Controller Area Network (CAN) Bus layer
- Counter-Timer
- Digital I/O
- Power-Conversion layer

Chapter 2 details the configuration and operation of the cube's Core Module. Chapter 4 details the behavior and architecture of the cube's Core Module. Detailed information on the hardware layers is found in the layer-specific manuals and on the website (www.ueidaq.com)

Chapter 2 Installation and Configuration

Installation consists of:

- PowerDNA software package installation
- Cube hardware setup
- Configuration

2.1 Initial Installation - Overview

This section outlines the steps to be taken in Section 2.2.

- STEP 1:** Install the PowerDNA software suite. The latest software suite can be found online at www.ueidaq.com/download; a copy is also included on the CD.
- STEP 2:** Connect the serial cable: from Cube RS-232 port to the host computer serial port
- a. Start a TTY client:

```
Start >> Programs >> UEI >> PowerDNA >> MTTY
```
 - b. Change the Baud rate to 57600 and Click Connect.
- STEP 3:** Connect the power supply to the Cube.
- STEP 4:** The Cube comes pre-configured with an IP address. Using MTTY, type [Enter] to test the prompt, for Coldfire: DQ> for PPC: =>. Then type:
DQ> show
- ```
ip: 192.168.100.2
netmask: 255.255.255.0
```
- STEP 5:** (optional) The recommended method of connection to the Cube is via a direct Ethernet cable connected to an external NIC. Connecting the cube directly to a LAN usually requires a change of IP address on the Cube. For example, your system administrator has assigned you the unused IP, 192.168.0.65. Here is how to change the IP to this example IP:

```
DQ> set ip // Sets this Cube's IP to 192.168.1.10
192.168.0.65 // Saves the newly changed configuration
DQ> store // Reboots the cube for the new IP to take
DQ> reset // effect
```

To make sure that the PowerDNA Cube is alive, ping it:

```
C:\> ping -n 1 192.168.0.65
```

- STEP 6:** Use PowerDNA Explorer for graphical configuration (see Chapter 3).

## 2.2 Initial Installation – Start-to-finish Guide

This section reviews how to perform an initial hardware and software setup when you first receive a PowerDNA Cube.

### 2.2.1 Inspect the package

Inspect the contents of the shipping package. With a standard PowerDNA Cube, you should find:

- The PowerDNA Cube itself, preinstalled with your selection of I/O Layers.
- The DNA-PSU-24 universal powerline brick, which plugs into an outlet and provides 24V dc output. The supply comes with a plug for the mains, an adapter cable ending in a Molex connector for plugging into the DNA Cube, and a daisy chaining cable for supplying additional Cubes with power from the same supply (max. of three Cubes total).
- Serial cable for initial hardware configuration and firmware downloading.
- CD-ROM with support software

### 2.2.2 Install Software

This section describes how to load the PowerDNA software suite onto a Windows- or Linux-based computer and run some initial tests.

The latest PowerDNA support software is online at [www.ueidaq.com/download](http://www.ueidaq.com/download); a known working copy is also on the PowerDNA Software Suite CD.

#### A. Software Install: Windows 9x/2000/XP

The PowerDNA CD provides two installers:

- PowerDNA Software Suite: low-level driver and PowerDNA tools
- UEIDAQ Framework: high-level programming examples (optional)

Both installers automatically search for third-party IDE and testing suites, and add themselves as tools to the found suites. Install third-party applications (e.g., LabVIEW, MsVS2003) before installing the PowerDNA Software Suite or UEIDAQ Framework.

To install the PowerDNA Software Suite, do the following:

**STEP 1:** Log in as Administrator.

**STEP 2:** Run Setup.

- a. Insert the PowerDNA Software Suite CD into your CD-ROM drive. Windows should automatically start the PowerDNA Setup program. An installer with the UEI logo and then PowerDNA Welcome screen should appear. If none appears, run setup.exe from the CD drive:

**Start >> Run >> d:\setup.exe >> OK.**

If you downloaded the most recent executable from [www.ueidaq.com](http://www.ueidaq.com), double-click to run the executable.

- b. Choose the PowerDNA Software Suite option.
- c. Unless you are an expert user and have specific requirements, we advise you to select Typical installation and accept the default configuration. The Software Suite installer requires and automatically

installs Sun's Java VM (JRE) for you, in addition to the full complement of tools. As an alternative, use the custom option to display and ensure that all of the packages necessary are installed.

- d. Companion Documentation:  
Quick Start Guide, Configuration & Core Module,  
I/O Layer Manuals, Low-level Programming Guide
- e. SDK: includes/lib for C/Java, examples, and Sun's JRE;  
(The SDK is not the UeiDaq Framework)
- f. PowerDNA Apps: PowerDNA Explorer, MTTTY
- g. PowerDNA Components (incl. DLL files)
- h. PowerDNA Firmware
- i. Click Next to continue through the dialogs.
- j. Click Finish to complete installation; restart the computer.

This Software Suite installed the bare-minimum tools needed in later steps: MTTTY, PowerDNA Explorer, and the low-level driver.

UEIDAQ Framework provides the structure for developing applications under C/C++, C#, VB.NET, ActiveX (VB6, Delphi), MATLAB, LabVIEW, DASyLab, LabWindows/CVI, OPC, and other programming languages.

**NOTE:** Because the installation process modifies your Windows registry, you should always install or uninstall the software using the appropriate utilities. Never remove PowerDNA software from your PC directly by deleting individual files; always use the Windows Control Panel/Add-Remove Programs utility.

### B. Software Install: Linux

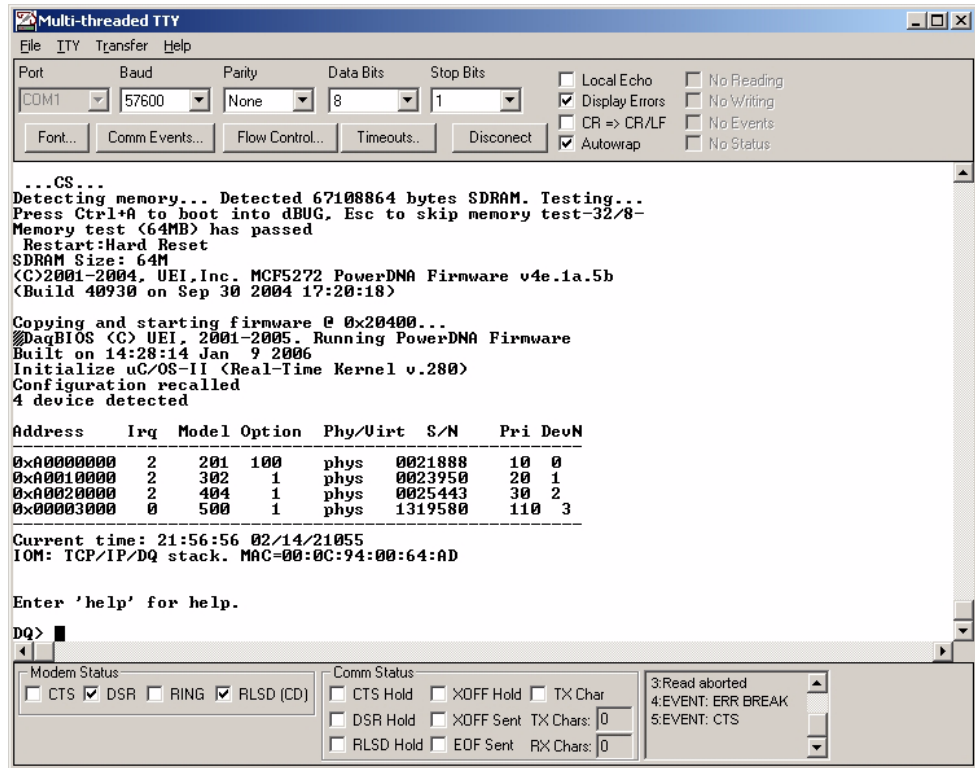
Linux: The PowerDNA\_\*.tgz file in the CD\Linux folder contains the software package for Linux. To extract the file to a local directory, enter:

```
tar -xjvf /path/to/powerdna*.tgz
```

Follow the instructions in the readme.txt contained therein.

**2.3.3 Initial Boot-up** This procedure is needed to prepare for network configuration. Do the following steps:

- STEP 1:** Familiarize yourself with front-panel layout. Note that all connections are made on front of the unit; no rear access is required in a rack-mounted configuration).
- STEP 2:** Attach the serial cable to the host PC and to the DNA Cube RS-232 port.
  - a. Run a terminal-emulation program (MTTTY) on the PC. Any terminal-emulation program may be used (MTTTY, Minicom, TeraTerm, etc.) Note that HyperTerminal probably will not work with a PowerDNA Cube.
  - b. Verify that COM parameters are set: 57600 baud, 8 bits, no parity, 1 stop bit.
  - c. Click Connect in MTTTY, or use the commands on one of the other terminal-emulation programs to establish communication with the Cube.
- STEP 3:** Power up the Cube (9-36V DC) by attaching the Molex-type power connector leading from the bundled DNA-PSU-24, a user-supplied source, or a daisychained line from another PowerDNA Cube. Note that the DNA-PSU-24 plugs into a 100-240V, 50/60-Hz outlet. Also note that the Cube does not have an On/Off switch.
- STEP 4:** As soon as the Cube powers up, it runs through self-diagnostic mode and generates output on the terminal program. A typical readout might be:



**Figure 2-1. Typical MTTTY Startup Screen**



The boot process displays the model, serial number, and position of layers. Type show <CR> to display information about cube configuration:

```
DQ> show
 name: "IOM_1234"
 model: 0x1005
 serial: 0012345
 mac:
00:00:11:AA:BB:CC
 fwct: 1.2.0.0
 srv: 192.168.100.3
 ip: 192.168.100.2
 gateway:
192.168.100.1
 netmask:
255.255.255.0
 udp: 6334
// IOM or I/O Module - is another
// name for a Cube
// Core Module > Model Number (1005:
// ColdFire)
// Core Module > Serial Number (S/N)
// of Cube
//Core Module > NIC Layer > MAC
// Address
// Define Cube procedure on startup
// IP Address of firmware server
// IP Address of this Cube
// IP Address of gateway
// IP Subnet Mask of this Cube
// UDP Port to receive commands on
```

All parameters can be changed; most notably, the cube's configured IP, gateway, and subnet mask (netmask).

### 2.3.4 IP Addresses on the PowerDNA Cube

The PowerDNA Cube ships with a preconfigured factory default IP address in nonvolatile memory (usually 192.168.100.2). This is a static IP address; the PowerDNA Cube never retrieves its IP address from a DHCP server. This section describes why and how to change the default IP address.

#### ***Should you change the IP?***

Yes, if you plan to use the Cube on a LAN where.

- High sampling rate is not necessary
- Some samples may be dropped due to network congestion and collisions
- The cube should be accessible by multiple parties on the LAN
- Multiple Cubes operate (and interact) on the same network

Alternatively, if you plan to use the Cube for high-speed measurements where reliability is necessary – a direct connection between the host PC and a NIC<sup>1</sup> is recommended. For a direct connection, see the following section, “Improving Network Performance”

#### ***How to change the IP.***

Both PowerDNA Explorer and a terminal-emulation program can change the IP. Consult your system or network administrator to obtain an unused IP. Let's say, for example, that your system administrator assigns you the IP 192.168.0.65.

To change the IP from the terminal program, enter the following commands:

---

1. NIC - Network Interface Controller; a commercially available Ethernet (i.e. IEEE 802.3) adapter.

```
DQ> set ip
192.168.0.65
Enter user password >
powerdna
DQ> store
DQ> reset
```

```
// Sets this Cube's IP to 192.168.0.65
// The default password is "powerdna"
// Saves the newly changed configuration
// Reboots the cube for the new IP to
//take effect
```

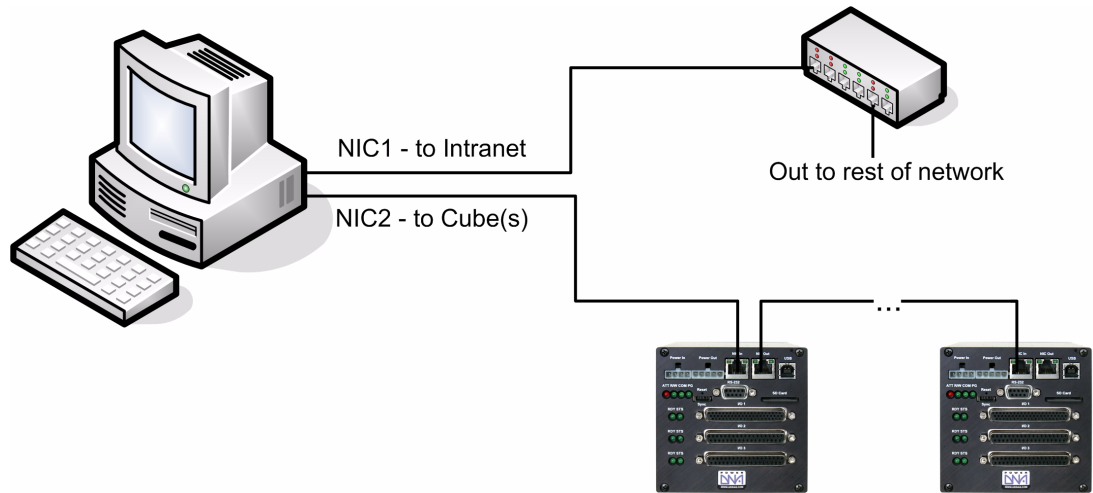
You can set any parameters listed with the “show” command in this manner. Connect the PowerDNA Cube to your switch with the bundled CAT5e cable.

If you can establish communications with a Cube, but later want to modify the IP address, you can also do so from within PowerDNA Explorer. After the exploratory process, go to the field where the application displays the IP address. Enter the new IP address and then hit <Return>. This action downloads the new IP address into the Cube’s non-volatile memory. You might also need to change the gateway and network mask to match settings on your LAN.

### 2.3.5 Improving Network Performance

To improve PowerDNA network performance, we recommend that you use a separate commercially available network interface controller (NIC) card and set up a dedicated mini-network for PowerDNA.

The goal of this section is to facilitate creation of such a network:



For example, assume that your office uses a Class C network (the class intended for small networks with fewer than 256 devices) and your host is configured with a static IP or via DHCP—Dynamic Host Configuration Protocol — a protocol for assigning dynamic IP addresses to devices on a network.

**STEP 1:** Obtain your networking configuration by using the Command Prompt:

*Start>>Programs>>( Accessories>>) Command Prompt*

```
C:\> ipconfig
```

```
Ethernet adapter NIC1 - Local Area Connection:
Connection-specific DNS Suffix . :
IP Address. : 192.168.1.10
Subnet Mask : 255.255.255.0
Default Gateway : 192.168.1.1
```

Linux users can use the more verbose “ifconfig” command instead.

Here, the subnet range 192.168.1.0-192.168.1.255 is being used by NIC1.

IP Addressing:

The range of usable addresses is defined by the IP address and subnet mask. An IP address is a number that is split into the range of 0.0.0.0 and 255.255.255.255. Here, the IP address is 192.168.1.10.

The subnet mask indicates where an address stops. For example, a subnet mask 255.255.255.240 has 15 usable addresses (255.255.255.255 - 255.255.255.240).

Here, the subnet is 255.255.255.0, or 255 addresses.

The subnet limits from anything.anything.anything.0 up to the max. The usable range for 192.168.1.10/255.255.255.0 is 192.168.1.1 to 192.168.1.254 (192.168.1.0 and 192.168.1.255 are reserved for Router and Broadcast messages).

The usable range for 192.168.0.4/255.255.0.0 is 192.168.0.1 to 192.168.255.255

The usable range for 192.168.100.2/255.255.255.0 is 192.168.100.1 to 192.168.100.254

---

Not every IP address from 0.0.0.0 to 255.255.255.255 is usable; however, these three ranges of IP addresses are guaranteed open for private use:

10.0.0.0 - 10.255.255.255  
 172.16.0.0 - 172.31.255.255  
 192.168.0.0 - 192.168.255.255

You need not use the entire set.

**STEP 2:** Install the secondary NIC card.

**STEP 3:** Set up a network that does not overlap the existing one.

The address space 192.168.1.0 – 192.168.1.255 is used. The IP address block, 192.168.2.1 – 192.168.2.255 is available and in the private range.

Let us choose 192.168.100.1 – 192.168.100.255 for the PC’s secondary NIC:

```
IP: 192.168.100.3
Netmask: 255.255.255.0
Gateway:192.168.100.3
```

Using the Network (Connections) in the control panel:

*Start >> Programs >> Control Panel >>Network (Connections)*

Right-click the adapter to bring up the properties window.

Open the TCP/IP properties of the adapter and edit to your liking.

**NOTE:** Refer to the Appendix at the end of this document: *“Configuring a Second Ethernet Card”* for step-by-step instructions on how to do this.

**STEP 4:** Confirm the network configuration at the Command Prompt:

*Start >> Programs >> (Accessories >> ) Command Prompt*

```
C:\> ipconfig
```

```
Ethernet adapter NIC1 - Local Area Connection:
```

```
Connection-specific DNS Suffix . . :
IP Address. : 192.168.1.10
Subnet Mask : 255.255.255.0
Default Gateway : 192.168.1.1
```

```
Ethernet adapter NIC2 - Local Area Connection 2:
```

```
Connection-specific DNS Suffix . . :
IP Address. : 192.168.100.3
Subnet Mask : 255.255.255.0
Default Gateway : 192.168.100.3
```

**STEP 5:** Set up the PowerDNA Cube to use the same subnet, namely:

```
Cube IP: 192.168.100.2 // this is the factory default
Gateway:192.168.100.3
Netmask: 255.255.255.0
```

To do this from a serial terminal-emulation program, enter the following commands when you get the DQ command prompt:

```
DQ> set ip
192.168.100.2

DQ> set gateway
192.168.100.3

DQ> set netmask
255.255.255.0

DQ> store

DQ> reset
```

```
// Sets this Cube's IP address to
// 192.168.100.2
// Sets this Cube's Gateway to
// 192.168.100.3
// Sets the subnet mask to 255.255.255.0

// Saves the newly changed configuration
// Reboots the cube for the new IP to
take
// effect.
```

**STEP 6:** Connect the PowerDNA Cube to your PC's second NIC, using the bundled CAT5 cable. The green lights should light up (try the other port, otherwise).

**STEP 7:** Ping the cube to make sure that it is alive.

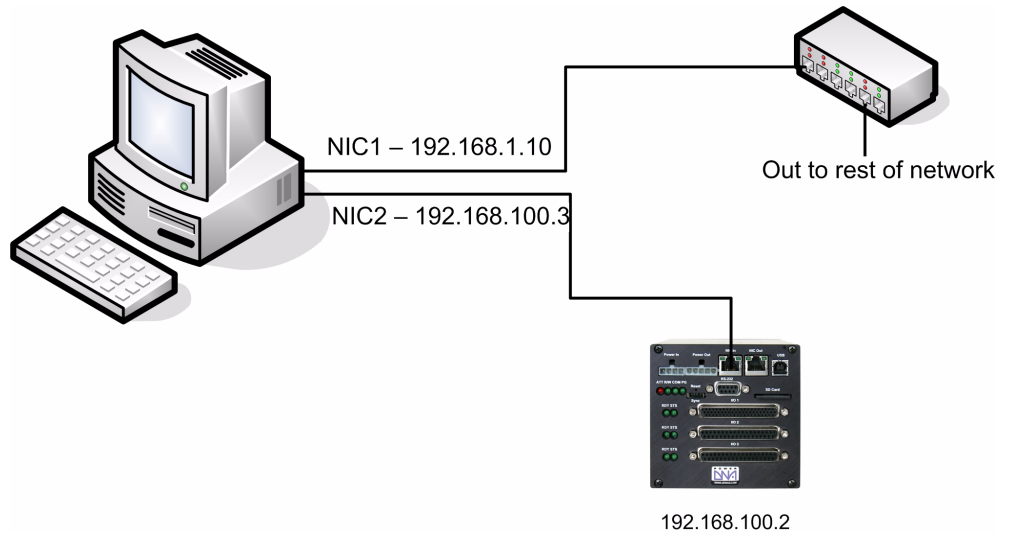
```
C:\> ping -n 1 192.168.100.2
Pinging 192.168.100.2 with 32 bytes of data:

Reply from 192.168.100.2: bytes=32 time<1ms TTL=128

Ping statistics for 192.168.100.2:
Packets: Sent = 1, Received = 1, Lost = 0 (0% loss),
```

The above is a successful response. A “Request Timed Out” message indicates error.

**STEP 8:** The Cube should now be configured as follows, where NIC1 uses a straight-through, and NIC2 uses a cross-over cable to the **NIC In** (or a straight-through cable will connect to **NIC Out**).



**STEP 9:** You may now use PowerDNA Explorer to access the cube. See Chapter 3.

### 2.3.5.1 Troubleshooting

The following checklist may assist you in troubleshooting a Cube.

- The PG (Power Good) LED is on: the Cube plugged in using 9-36V DC.
- The green lights on NIC In or NIC Out are blinking: CAT5e cable is connected
- Use the command prompt to ping <cube IP> (e.g., ping 192.168.100.2)
  - a. Disable (temporarily) the firewall on the secondary NIC.
  - b. Check the secondary NIC network settings.
  - c. Check the cube's network settings.

Use MTTY and hit Connect.  
 Press [Enter] to display the DQ> or => prompt.  
 (No prompt indicates that you are not connected)  
 Verify that the serial cable is firmly connected to the RS-232 port.

Verify the settings: 57600 baud, no parity, 8 data bits, 1 stop bit. Try COM1, COM2, COM3 then hit Connect and press [Enter].

- Reboot the cube. The start-up screen should display upon restart.
- If all else fails, contact UEI support at: [support@ueidaq.com](mailto:support@ueidaq.com)
- Type “show” to verify the Cube’s IP, Subnet Mask, and Gateway
- Ensure that the computers are on a valid subnet and have valid IPs
- Finally – contact UEI for support at: [support@ueidaq.com](mailto:support@ueidaq.com)

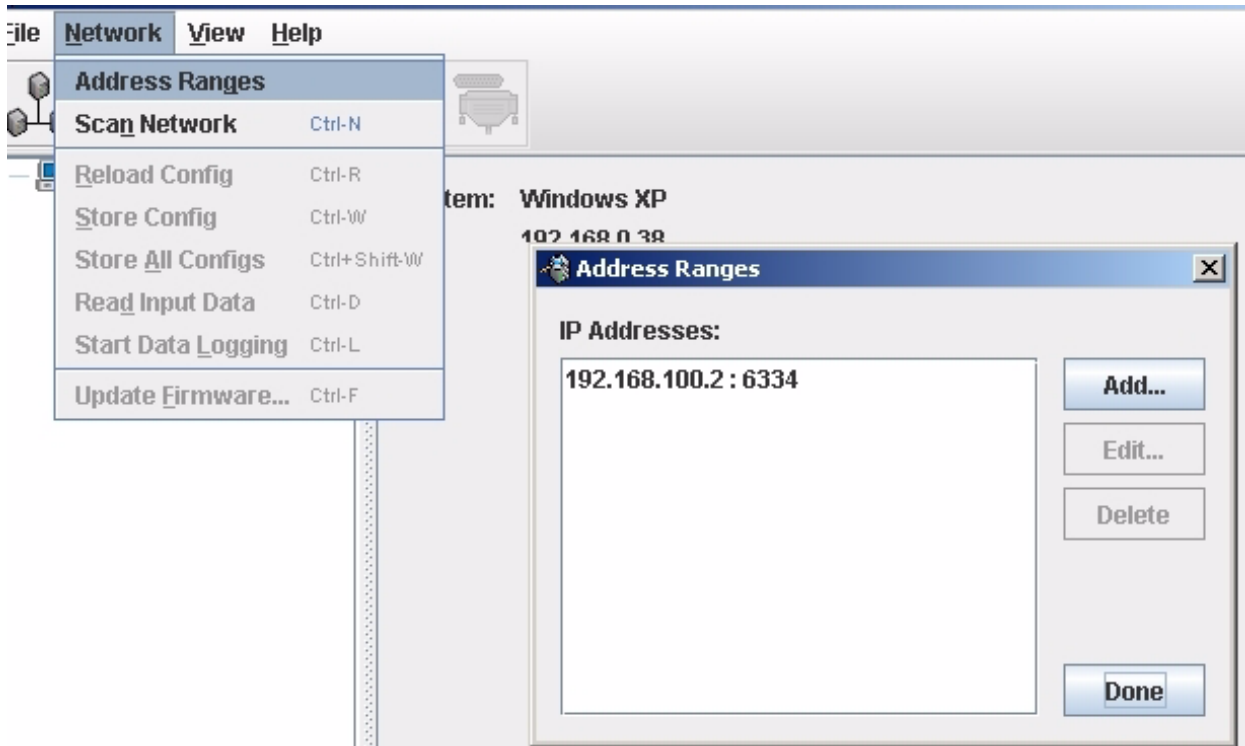
### 2.3.6 PowerDNA Explorer Quick-Start

PowerDNA Explorer does just what its name implies: it “explores” the LAN, looking for connected PowerDNA Cubes. Chapter 3 covers the PowerDNA Explorer in detail. This section/page only provides a quick-start guide.

The PowerDNA Explorer identifies PowerDNA Cubes on a selected network – the discovered Cubes are listed on the left-hand-side pane. Select a cube to display pertinent hardware and firmware information. Select a layer of a specific cube to manipulate its inputs or outputs. In brief, this useful tool lets you verify that the Cube is communicating with the host and that the I/O Layers are functioning properly.

To scan the network for PowerDNA Cubes, provide a set of addresses to scan. Do the following:

**STEP 1:** Select Network  Address Ranges from the menu:



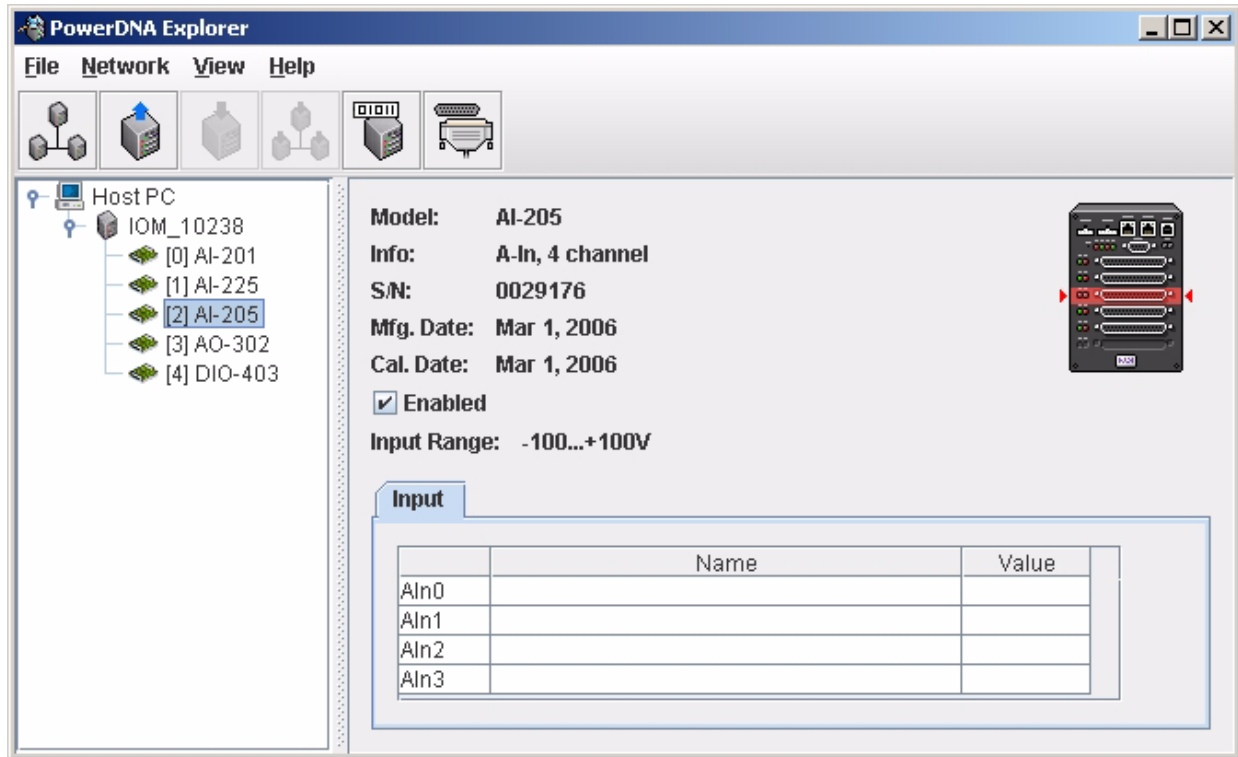
**Figure 2-2. PowerDNA Explorer Startup Screen**

**STEP 2:** Add the IP address of the PowerDNA Cube (e.g., 192.168.100.2); click Done.

**STEP 3:** Now scan the LAN for PowerDNA Cubes: *Network >> Scan Network*

One or more gray cube-like icons will display in the left-hand-side of the cube. If no cube icons are displayed, see the Troubleshooting note in the previous section.

**STEP 4:** Double-click a cube to see its information and list the layers:



The screenshot above is from the PowerDNA Explorer Demo. The “demo” is just a simulator for users without cubes – or for new users who want to explore the PowerDNA Explorer program without reading/writing to real hardware. Run this program and hover your mouse over the buttons to read the tool-tips and learn through interacting with the program.

Some quick notes:

- To use the layer, the “Enabled” check box should be set.
- To read from a layer, click the second-to-last button: “Read Input Data”
- To write to the layer, change the value and click the third (or fourth) button with the red arrow on top of the cube: “Store Configuration”. The cube with the blue arrow above it restores the configuration.
- To change the IP, change the number, deselect the field, and “Store Configuration”. Take care not to set the IP Address to outside of the network’s configuration subnet -or- to an IP address that is currently in use, as the cube will then become unreachable.

See Chapter 3, PowerDNA Explorer, for additional information and instruction.

## 2.4.7 Updating Firmware

Firmware in a PowerDNA Cube's CPU layer stores configuration data, along with a user application (user-app is compiled on a host PC).

Updated firmware is periodically released to introduce new features and to improve the performance of existing features. Updated releases of the firmware are bundled with the entire PowerDNA Software Suite, available for download at any time from the UEI web site ([www.ueidaq.com](http://www.ueidaq.com)).



### **CAUTION!**

***If you update the firmware in a Cube, be sure to use the PDNA Explorer from the same release as that new firmware.***

After installing the PowerDNA Software Suite, browse to the installation's Firmware directory (e.g. C:\Program Files\UEI\PowerDNA\Firmware).

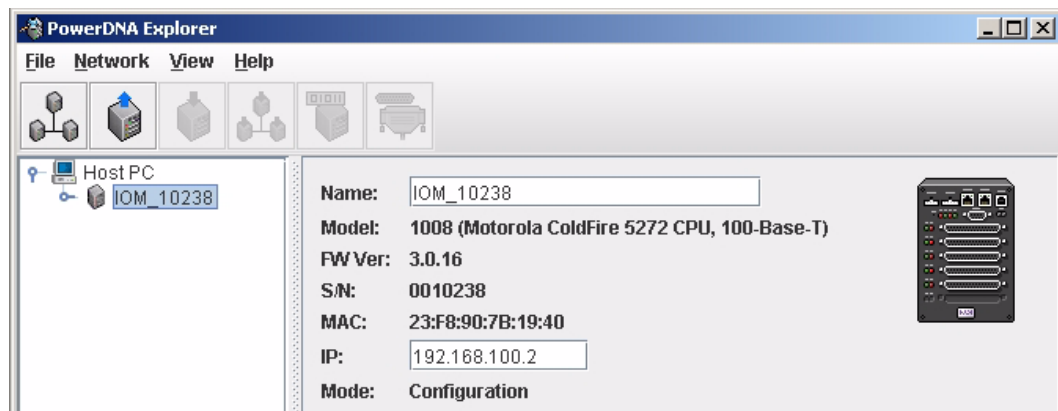
The directory may contain MTTTY, updated firmware installation instructions "FirmwareInstall.html," and two sub-directories containing the firmware. Choose the sub-directory corresponding to the architecture of your cube: ColdFire (CF/CM) with extension S19, or PowerPC (PPC), with extension MOT.

### ***Determining the version of your PowerDNA cube with PDNA Explorer:***



***Before updating the firmware of a PowerDNA cube, check the cube version to determine which update method to use.***

- a. Supply power to the PowerDNA cube.
- b. Connect the PowerDNA cube to its network.
- c. Start PowerDNA Explorer on the Microsoft Windows desktop from  
*Start >> Programs >> UEI >> PowerDNA >> PowerDNA Explorer*
- d. Choose Network > Scan Network
- e. Select the PowerDNA cube you wish to query (by clicking the cube).
- f. The version is given in the FW Ver field.





If the FW Ver field has is version 2.x.x, or 3.x.x (let x be any version number), you should follow the *Firmware Update Instructions [CM5, CM8]* section below. For other versions of firmware (e.g., 1.x.x), refer to the user manual on the CD that accompanied your device when you purchased it.

### 2.4.8 Firmware Update Instructions

Before using a new release of the libraries and applications to communicate with your PowerDNA cube, you must install the latest version of the firmware on the PowerDNA cube. The version of the firmware *must* correspond to the version of the PowerDNA Software Suite — mismatched versions cause an error.

Instructions for updating the PowerDNA Cube via PowerDNA Explorer (over an Ethernet LAN line), and over an MTTTY (serial line) follow.

To upload firmware with PowerDNA Explorer over LAN, do the following:

- STEP 1:** Supply power to the PowerDNA cube.
- STEP 2:** Connect the PowerDNA cube to its network.
- STEP 3:** Start PowerDNA Explorer on the Microsoft Windows desktop from *Start >> Programs >> UEI >> PowerDNA >> PowerDNA Explorer*
- STEP 4:** Choose *Network >> Scan Network*
- STEP 5:** Select the PowerDNA cube to be updated.
- STEP 6:** Select *Network >> Update Firmware...* from the menu.



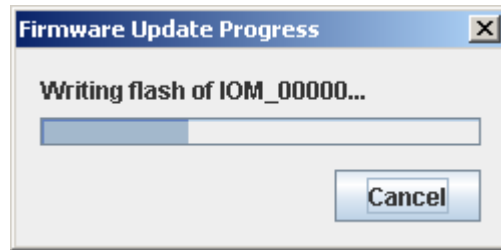
**Figure 2-3. Update Firmware Menu Item**

- STEP 7:** Click on “Yes” when you see the prompt:  
 “Are you sure you want to update firmware...”
- STEP 8:** Double-click on the dq\_ram.S19 file.
- STEP 9:** Enter the password to continue. More information about passwords can be found in the “*Interfacing to the CM module using a serial interface*” section of this manual. PowerDNA cubes come with the default password set to `powerdna`.



**Figure 2-4. Password Dialog Box**

**STEP 10:** Wait for the progress dialog to complete. The PowerDNA cube will then be updated and running the new firmware.



**Figure 2-5. Firmware Update Progress Dialog Box**

Each cube is updated in three steps. First, the firmware is transferred to the cube. Second, the firmware is written to the flash memory. During this step, the R/W light on the front of the cube is lit, in addition to the PG light. Third, the cube is reset. During this step, the ATT, COM, and PG lights are lit, and the R/W light will turn on and off periodically. When the cube is finished resetting, only the PG light is lit.

To upload firmware over serial port using a terminal client (MTTTY, do the following):

#### **Under DNA-CM5 and DNA-CM8:**

- STEP 1:** Establish communications between the PC and a Cube over the serial link.
- STEP 2:** Press the Hardware Reset switch on the front of the Cube to reset the CPU Layer.
- STEP 3:** While the Cube is starting up again, press <Ctrl>+<A> to activate the download screen (indicated by a #> prompt).  
If you get to the DQ> prompt, you waited too long and must return to Step 2.
- STEP 4:** Enter the dl command to enter the firmware-download routine.
- STEP 5:** Transfer the file. Depending on which terminal-emulation package you decide to run, you usually initiate the download with a command similar to “send”. In MTTY, go to the top menu and select Transfer >>Send file (text). When it asks for a file, go to the PowerDNA\Firmware directory and select the .S19 or .MOT firmware file. The download procedure will take roughly a minute.
- STEP 6:** To tell the Cube to save the new firmware into EPROM, enter the commands
- ```
upuser <CR>
update
```
- STEP 7:** Enter go to complete the firmware-update procedure and return to the DQ> prompt.

Under DNA-PPC5 and DNA-PPC8:

- STEP 1:** Establish communications between the PC and a Cube over the serial link.
- STEP 2:** Press the hardware Reset switch on the front of the Cube to reset the CPU Layer, or type: reset all
- STEP 3:** While the Cube is starting up again, Press ESC to go into u_boot.
- STEP 4:** Type the command to erase firmware download area in the Flash memory:

```
=> erase all
=> loads romimage.mot// loads stores firmware into the flash while
// downloading it.
```

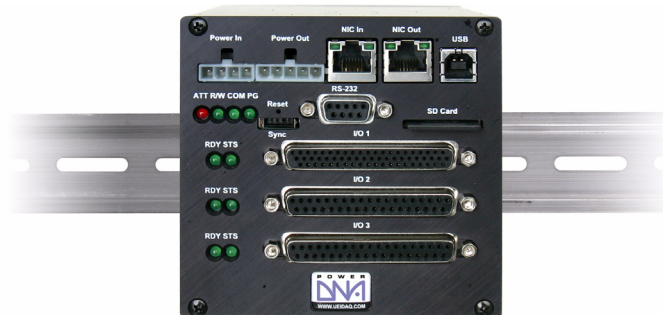
- STEP 5:** Transfer the Motorola firmware file. Use *Transfer » Send File*, and select `\Program Files\UE\PowerDNA\Firmware_PPC\romimage_3_x_y.mot`. A progress bar will appear in the lower left corner of MTTY indicating progress.
- STEP 6:** Wait for the upload to complete (it may take a few minutes).
- STEP 7:** After the process finishes, enter the `fwjmp` command. The PowerDNA cube will then be updated and running the new firmware. At this point, only the PG light on the cube remains lit.

2.5 Mounting and field connections

Mount the Cube directly to the application hardware either by screwing it directly to the machine or by using the optional DIN rail clip (DNA-DR). A normal DIN rail comes with screws you can use to mount the rail onto another surface or piece of equipment. However, because the Cubes are designed to fit into applications where space is at a premium, it may sometimes be difficult to attach the rail in this way. For such cases, we include a special adhesive tape for attaching the rail to any desired surface.



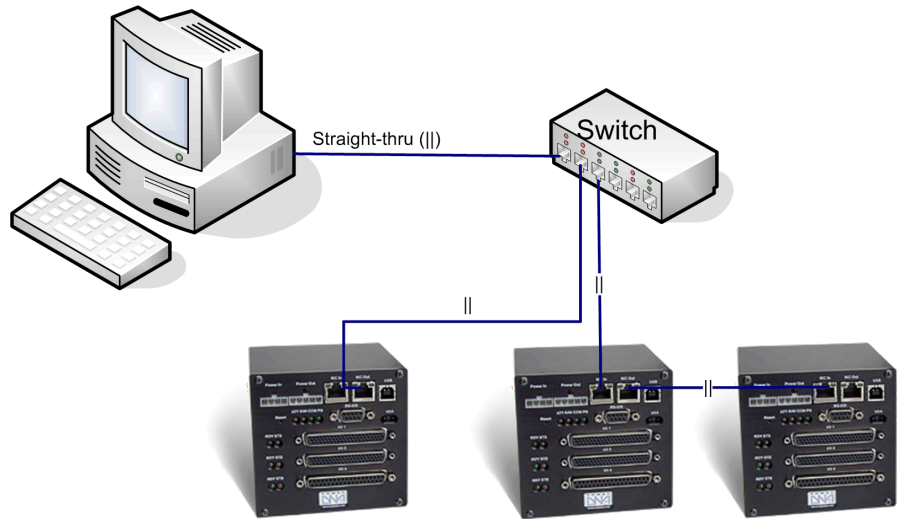
CAUTION! Take care when deciding on which surface you plan to mount a Cube. For example, using the adhesive strip, you can normally attach the DIN rail to a wall without causing any damage, as shown below — unless the wall has a sensitive coating such as delicate paint or wallpaper.



2.6 Wiring

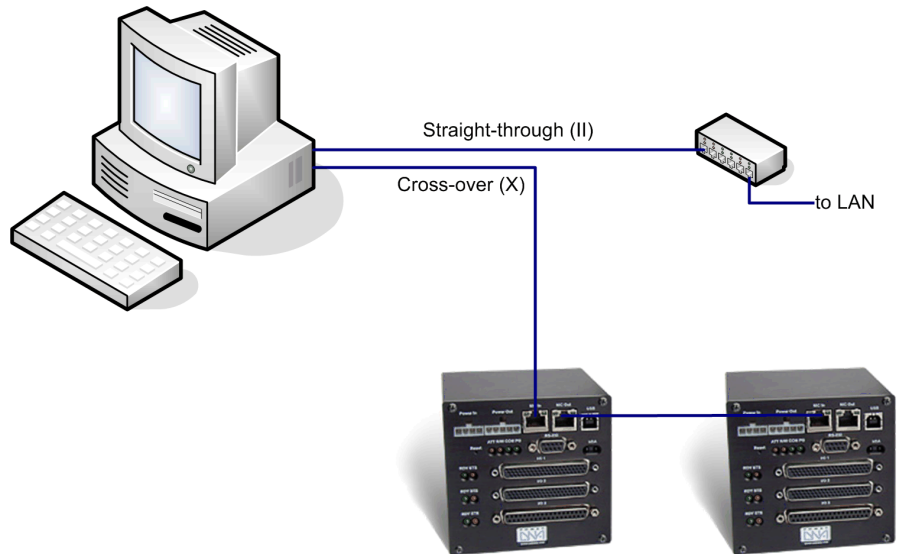
100BaseTX/100BaseFX Wiring Configurations

Typical wiring configurations for 100BaseTX/100BaseFX networks are shown in the following figures.

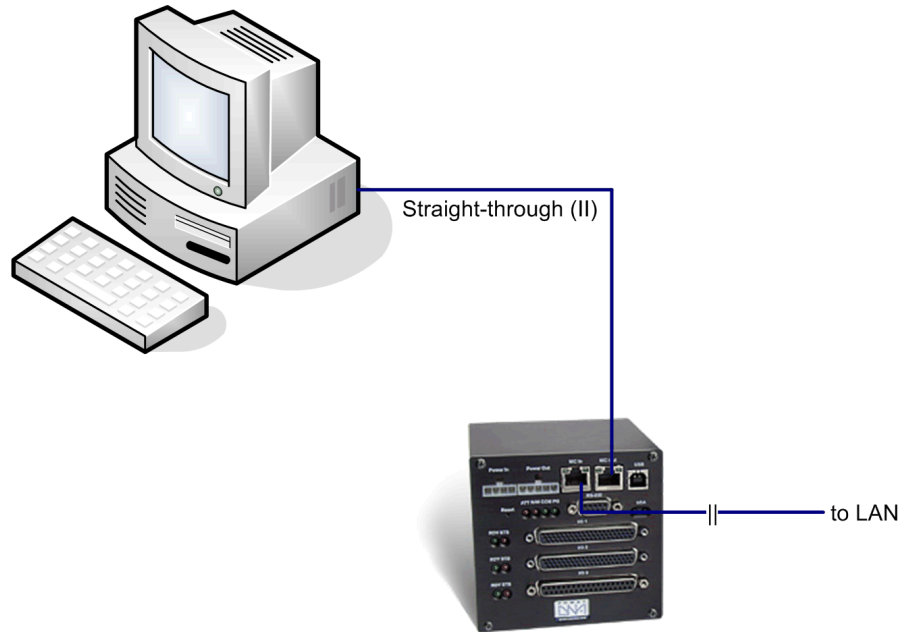


Cubes may also be connected with standard straight-through lines through a switch.

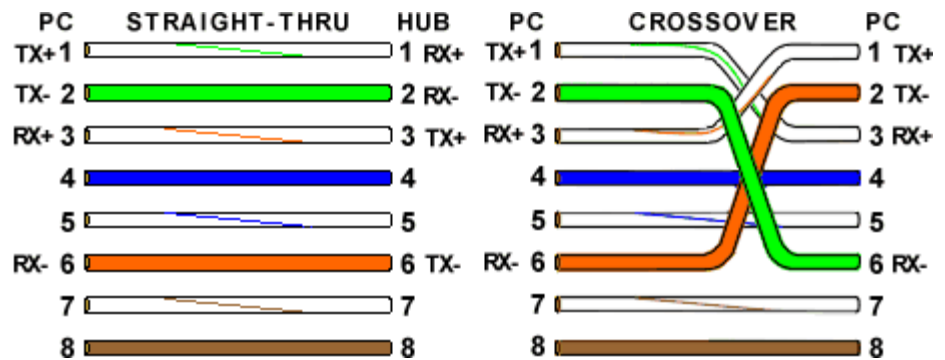
Alternatively, a cross-over cable may be used to directly connect to a cube, as shown below. This improves performance (and isolates the cube from problems with the switch).



For a fast connection in the field, you may connect a straight through cable to the NIC Out jack, as shown below. Use the NIC In jack to connect out to the LAN. The reason that this works is that in the NIC Out jack, the Rx/Tx lines are crossed over for you, so the wiring acts like a cross-over cable for you.

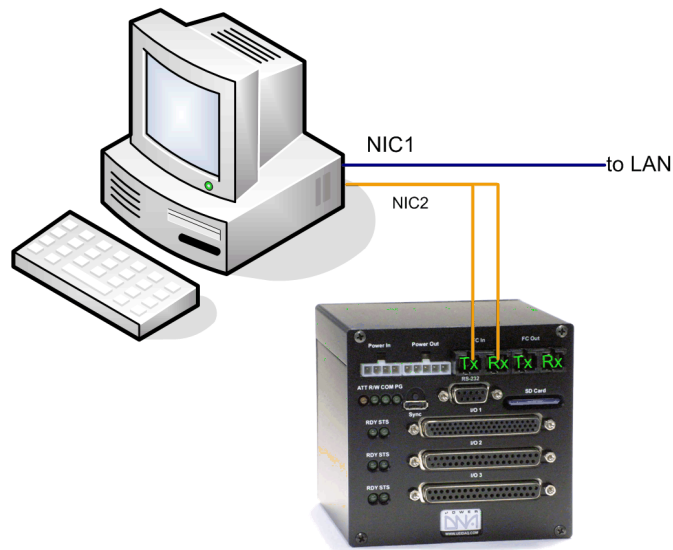


A crossover cable is the same as a straight-through except the Rx/Tx lines are inverted, as shown below:



NOTE: The above configurations work with CM and PPC (not FCM, or FPPC) when used in conjunction 8-wire Category 5 copper cabling (i.e. CAT5/ CAT5e) less than 100meters in length, and a 10/100Mbit NIC, or switch.

For FCM and FPPC cubes, use a fiber NIC, as shown below:



In this diagram, **NIC1** is a copper NIC connecting the PC to the LAN (optional). **NIC2** is an Intel network card in the PC used to connect to the cube's built-in fiber ports. A multi-mode optical cable with SC-type plugs like this one is used to connect to the Tx/Rx plugs. In 100Base-FX mode, the maximum transmission range (without a repeater) is 2km at full-duplex, or 400m at half-duplex. The cube uses an HFBR-5803 transmitter capable of communication at 100Mbps.

2.7 Peripheral Terminal Panel Wiring

Refer to the companion layer manuals for proper wiring to layers.

2.8 Repairing (and upgrading) Your Cube

PowerDNA Cubes come from the factory fully configured and calibrated. They are not suited for field upgrades or repairs. Should you encounter a problem with a Cube, or should you want to enhance or otherwise modify the selection of I/O layers in a Cube, you must send the unit back to the factory or to your local distributor. This process requires that you request an RMA number from UEI. To do so, you must provide the following information:

1. Model Number of the Cube
2. Serial Number of the Cube
3. Reason for return
 - Calibrating the layer(s)
 - Defective layer for repair
 - Upgrade with additional layer(s)

UEI will process the request and issue an RMA along with an estimate for the work involved to handle your request as well as the associated costs.

Chapter 3 PowerDNA Explorer

PowerDNA Explorer simplifies configuration and setup of a PowerDNA cube under Microsoft Windows.

This section describes the various menus in PowerDNA Explorer.

NOTE: The PowerDNA Explorer DEMO lets you safely explore the menus and layer screens without the need for using actual PowerDNA cubes.

3.1 The Main Window

The Main Window of the PowerDNA Explorer is shown in **Figure 3-1**.

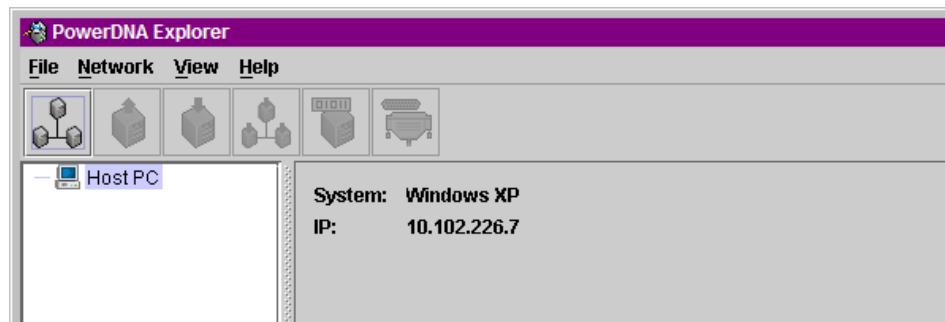


Figure 3-1. PowerDNA Explorer Main Window

The Main Window is the window you see when the PowerDNA Explorer is first launched and is where you do most of your work. It has four main parts: the Menu Bar, the Toolbar, the Device Tree, and the Settings panel.

3.2 Menu Bar

The Menu Bar contains the following menus and menu items.

3.2.1 File Menu

Preferences brings up the preferences dialog.

The preferences dialog allows you to specify the network timeout interval. This is the length of time PowerDNA Explorer will wait for response from a PowerDNA cube before giving up with an error. It defaults to 100 milliseconds.

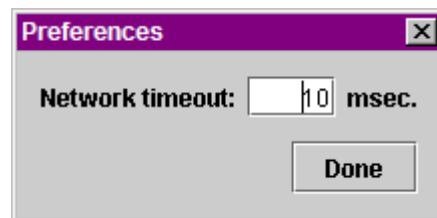


Figure 3-2. Preferences

Exit exits the application. If there are unsaved device settings changes, you are prompted for confirmation.

3.2.2 Network Menu *Address Ranges* brings up the Address Ranges dialog, allowing you to specify where to scan for devices.

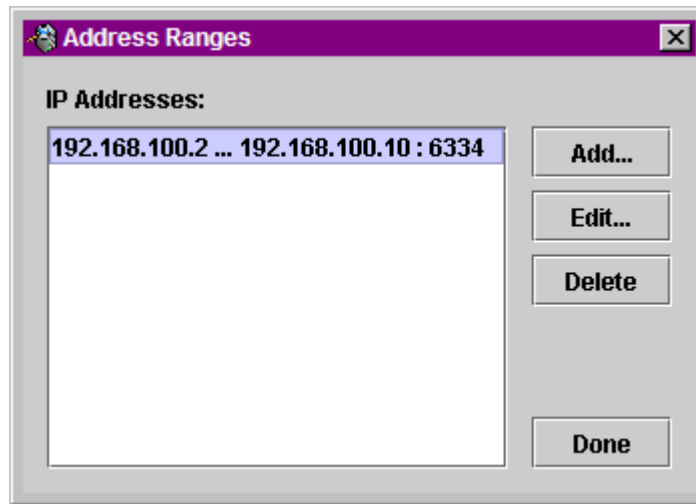


Figure 3-3. Address Ranges Dialog Box

The Address Ranges dialog allows you to specify the IP addresses and UDP port to use to find devices. You can specify individual addresses as well as address ranges. The specified items appear in a list that can be added to and deleted from. This list defaults to a single range item that specifies the range 192.168.100.2 ... 192.168.100.10.

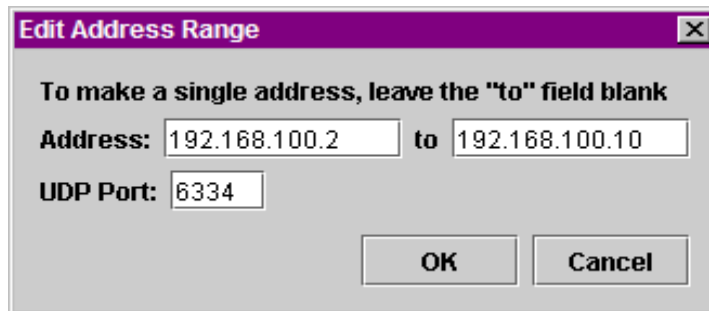


Figure 3-4. Edit Address Ranges Dialog Box

Scan Network scans the network for devices and populates the device tree. How much of the network is scanned depends on the settings in the Network Ranges dialog.

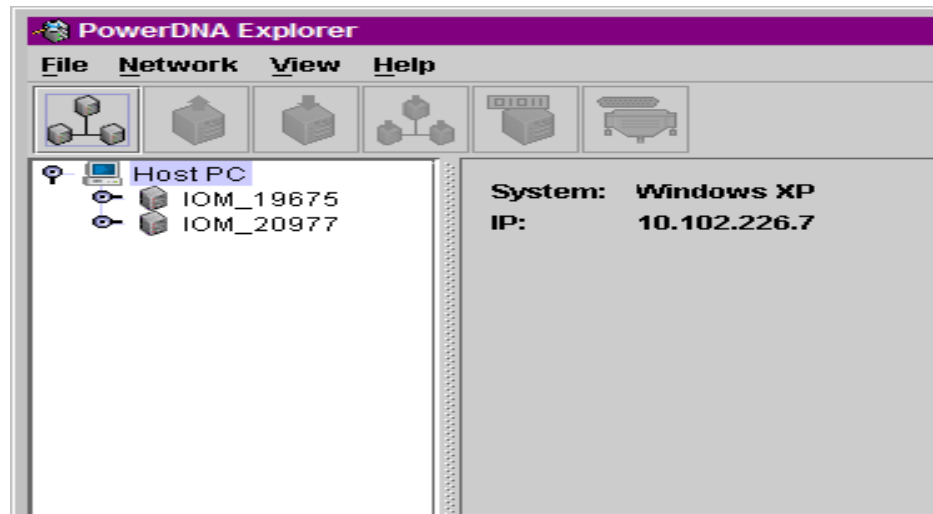


Figure 3-5 . After a Network >>Scan Network

If you choose *Scan Network* when the device tree is already populated, any new devices discovered will be added to the tree. Any existing devices that are missing will be removed from the tree, unless you have made unsaved changes to such a device's configuration, in which case it will be marked in the tree as missing.

Reload Config re-reads the configuration of the current device selected in the Device Tree. If you have made changes to the settings in the settings panel for the current device, Read will replace those settings with the device's current settings, after prompting for confirmation.

Store Config writes the currently selected device's changed settings to the device. The button is disabled for devices that haven't been modified.

Store All Configs writes all of the changed devices' settings to the devices. The button is disabled if no devices have been modified.

Read Input Data is enabled when the currently selected device is an input device layer. It reads the current input values to the device and causes them to be displayed in the settings panel.

Update Firmware... loads a firmware update file to all connected PowerDNA cubes if Host PC is selected. It updates only one PowerDNA cube when the specific PowerDNA cube is updated. More details about this can be found in the section *Updating firmware in a version 2.0 PowerDNA cube*.

Note that writing certain configuration changes to a PowerDNA cube running firmware 2.0.16 will bring up a password dialog box. More information about passwords can be found in the "*Interfacing to the CM module using a serial interface*" section of this manual. PowerDNA cubes come with the default password set to "**powerdna**".



Figure 3-6. Password dialog box for Store Config and Store All Configs



Figure 3-7. Password Dialog Box for Update Firmware . . .

3.2.3 View Menu

Show Wiring Diagram is a friendly reminder of the connector pins for a specific layer. All layers have this option, and we display this one as an example. The wiring diagrams in PowerDNA Explorer match the wiring diagrams in this manual in the sections for each layer.

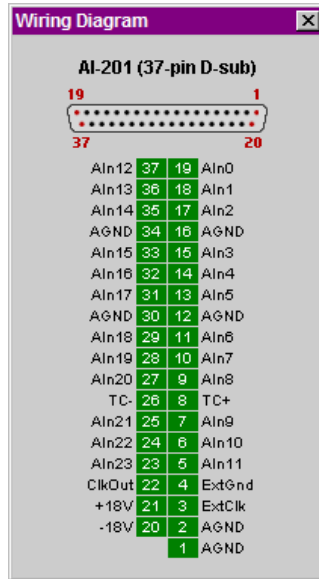


Figure 3-8. Example of a Wiring Diagram

3.2.4 Help Menu

About PowerDNA Explorer shows the **About ...** box, which shows the program icon, program name, version number, company name, and copyright notice.

3.2.5 Toolbar

The toolbar contains the following buttons: **Scan Network**, **Reload Config**, **Store Config**, **Store All Configs**, **Read Input Data**, and **Show Wiring Diagram**. They duplicate the functionality of the corresponding menu items as described above.

3.2.6 Device Tree

When the application is first launched, the tree contains just a root item representing the host computer. When you select **Scan Network** from the **Network** menu or the toolbar, the device tree gets populated with all central controllers, IOMs, and device layers accessible from the network, as filtered through the **Network Ranges** dialog. Central controllers, if any, appear as children of the **Host PC** item. IOMs that are connected to the PC without use of a central controller also appear as direct children of the **Host PC** item.

Each item has an icon indicating whether it is a central controller, IOM, or layer. The text label for each item is the device's model number, name, and serial number. Layers are also labeled with their layer number in parentheses.

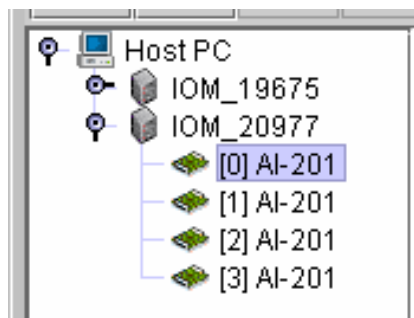


Figure 3-9. Example of the Device Tree

When an item is selected in the tree, the settings panel changes to reflect the settings for that device. The first time an item is selected, the device is queried as though you had invoked the Read command. On subsequent selections of the same item, the last settings are re-displayed. Thus, if you made changes but did not write them to the device, the changes are remembered. Invoking the Read command will re-read the device and overwrite the current settings in the settings panel.

Devices whose settings have changed, but have not been written are displayed in bold italics in the tree to provide a visual cue. Changed devices that become missing on a subsequent invocation of Scan Network turn red in the tree. (Unchanged items that become missing are simply removed from the tree.)

3.2.7 Settings Panel The settings panel presents a set of controls that allow you to change the settings of the device currently selected in the device tree.

3.2.7.1 IOM Settings The settings panel has the following controls when an IOM is selected in the tree.

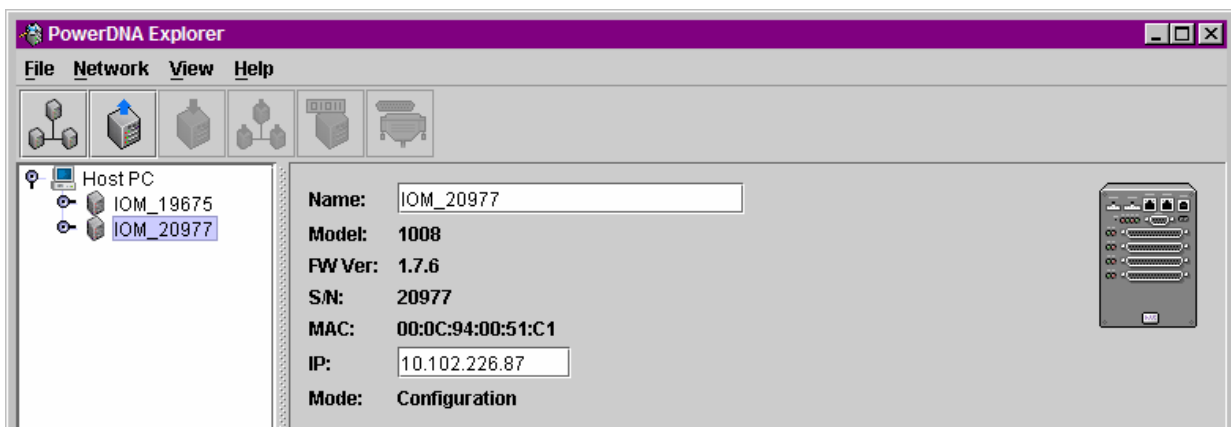


Figure 3-10.. Example of IOM Settings Panel for a PowerDNA cube

Name shows the IOM name. It can be changed.

Model shows the model number of the IOM.

FW Ver shows the version of the firmware installed on the PowerDNA cube.

S/N shows the serial number of the IOM.

MAC shows the MAC address. It cannot be changed, and thus is informational only.

IP Address shows the IP address of the IOM. It can be changed.

Mode shows the mode the PowerDNA cube is in: *Initialization*, *Configuration*, *Operation*, or *Shutdown*. These modes are described in the section, *IOM Modes*.

3.2.7.2 Device Layer Settings

Figure 3-1 shows the screen for displaying device layer settings.

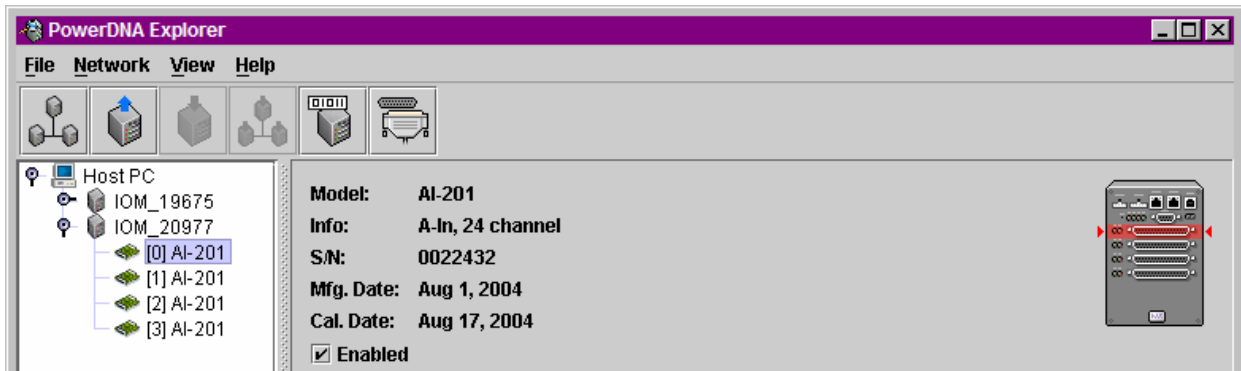


Figure 3-11 Example of Device Layer Settings for a Layer

Each layer has the following settings.

- **Model** shows the model number of the layer.
- **Info** shows some key features of the layer: A for analog, D for digital, In for input, Out for output, and a number of channels available.
- **S/N** shows the layer's serial number.
- **Mfg. Date** shows the manufacturing date of the layer.
- **Cal. Date** shows the date of the last calibration done to the layer.
- **Enabled** is a checkbox that, when unchecked, excludes the device from configuration. The device is excluded from the Store All Configs command, and the Reload Config command is disabled. Also, the device appears gray in the tree. All devices are enabled by default.

- Select *Network >>Read Input Data* to update the Value column of any layer, as shown below:

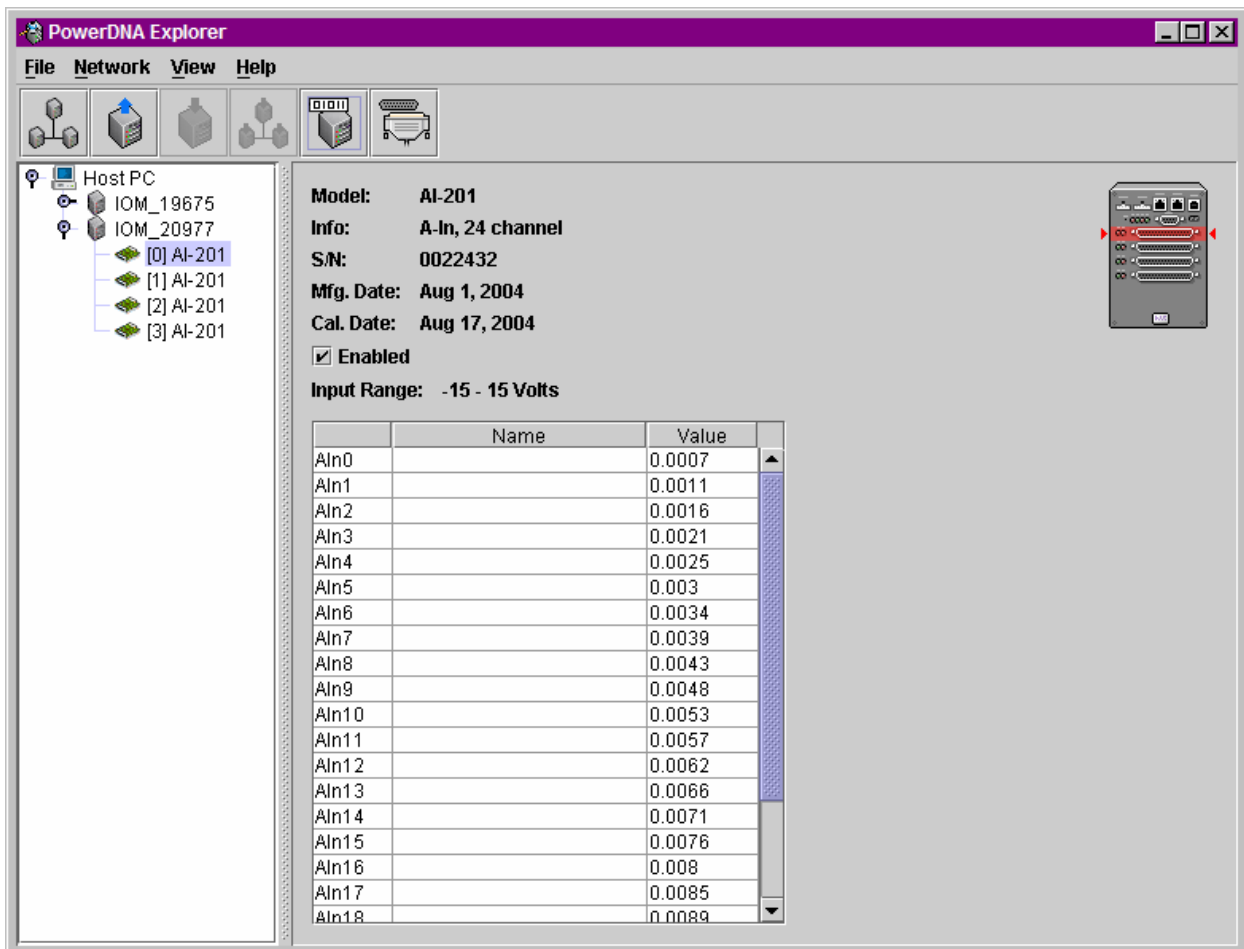


Figure 3-12. Screen from Network >> Read Input Data

At the screen shown above, you can add/edit channel names. After editing names, choose *Network >> Store Config* to save changes to the layer. This is true for all layers.

Also, if you have changed a configuration value, but have not chosen *Network >> Store Config* to save them, previous values can be re-read from the layer, using *Network >> Reload Config*.

AI-205 and AI-225 layer screens are same as the AI-201 layer, but with different input ranges and number of channels.

In addition, digital and analog output layers have settings specific to their layer types.

3.2.8 Digital Input/Output Layer Settings

We'll use the DIO-405 as an example to start with, then show how the DI-401, DO-402 and DIO-403 are different.

NOTE: Use *Network >> Read Input Data* to see immediate input values in Input tabs. Use *Network >> Store Config* to save values to the layer.

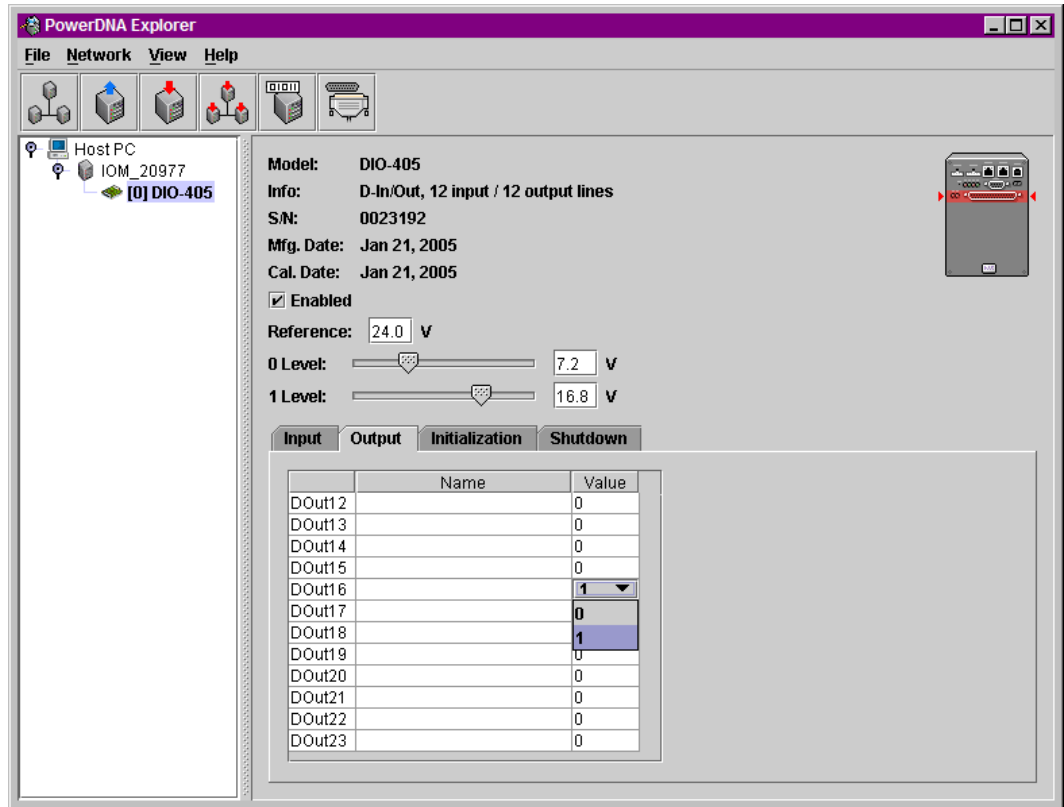


Figure 3-13. Example DIO-405 Layer Inputs

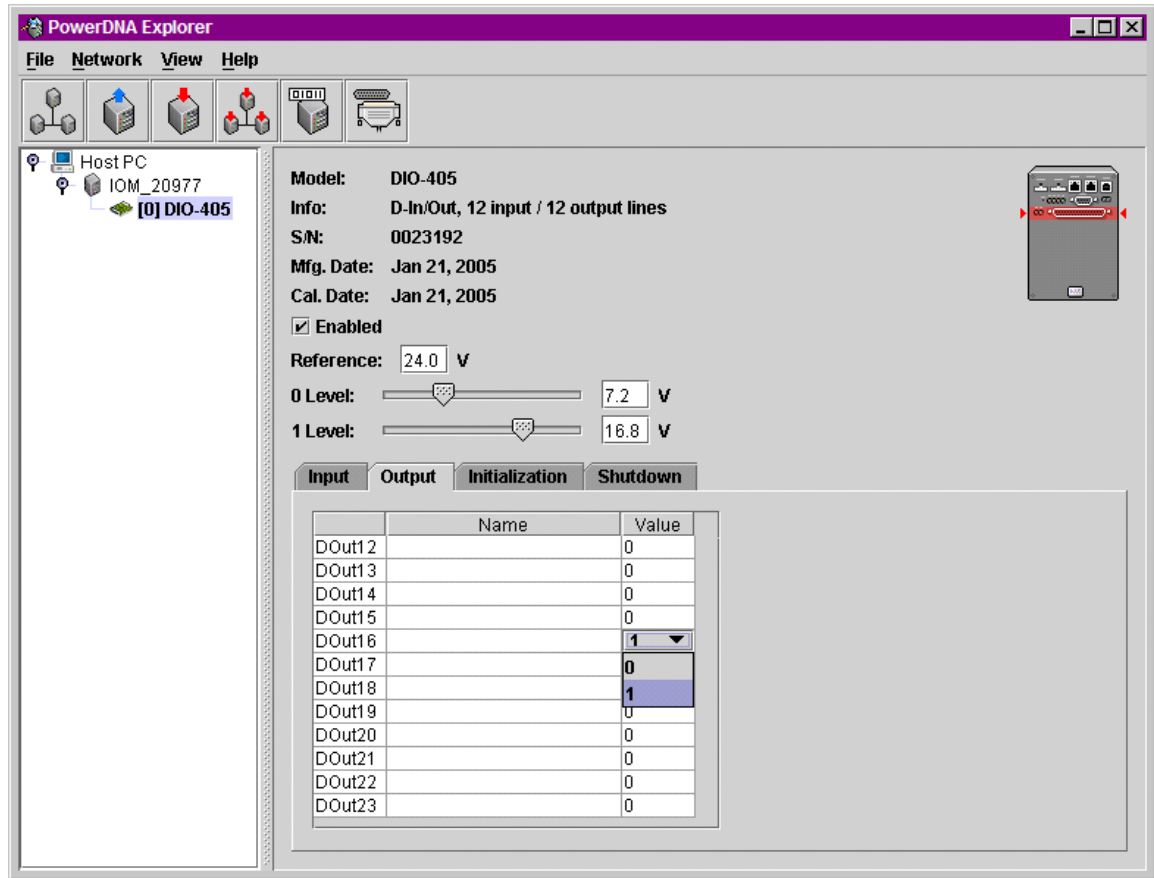


Figure 3-14. Example DIO-405 Layer Outputs

Reference is a reference voltage.

0 level/1 level are hysteresis values described fully in the DIO-401/2/5 section of this manual.

Input/Output/Initialization/Shutdown tabs switch between settings for init and shutdown states, as well as operation mode configuration, and display of current data.

All tabs contain the following columns:

- The unnamed first column contains the channels.
- **Name** is a user-defined string.
- **Value** contains 0 or 1. It is a drop-down menu for output channels allowing you to select 0 or 1.

The DI-401 layer just has Reference and 0 and 1 Level controls, and Input tab.

The DO-402 layer just has Output, Initialization, and Shutdown tabs; no Reference value or Level sliders.

The DIO-403 layer is different because it groups 8-bits at a time into ports, and three ports into two channels. For the sake of abstraction in PowerDNA Explorer, we'll call all the ports channels.

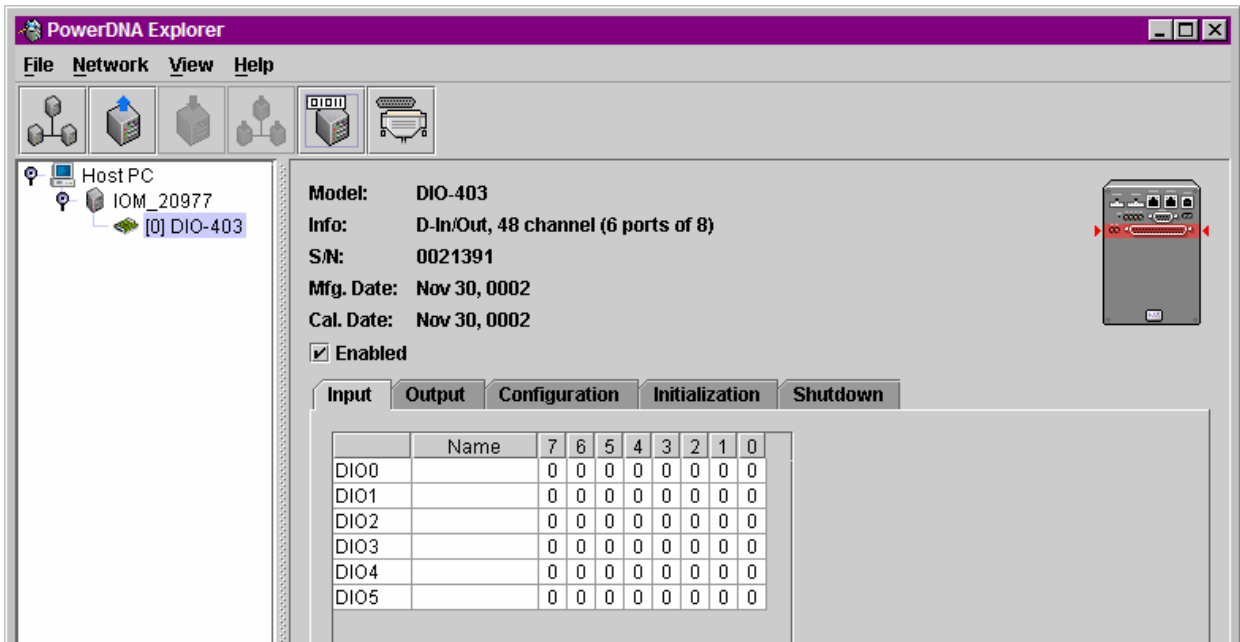


Figure 3-15. Example of DIO-403 Layer Inputs

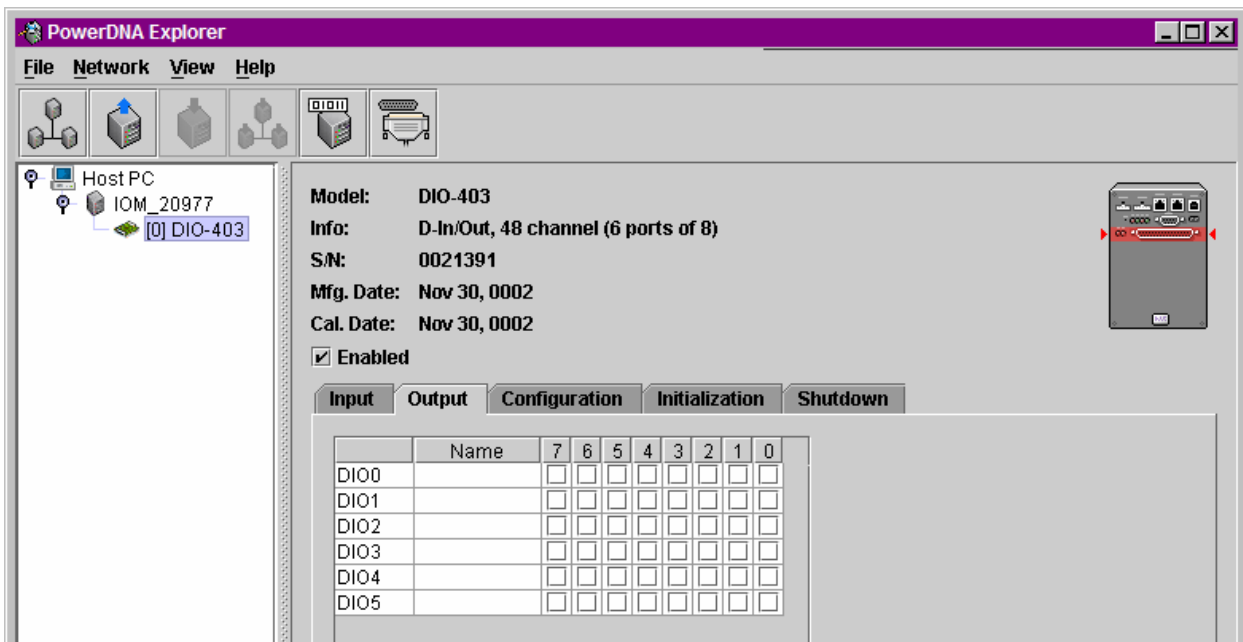


Figure 3-16. Example of DIO-403 Layer Outputs

Input/Output/Configuration/Initialization/Shutdown tabs switch between settings for init and shutdown states, as well as operation mode configuration, and display of current data.

Input/Output tabs get/set the current input/output values. They contain the following columns:

- The unnamed first column contains the channels.
- **Name** is a user-defined string.
- **7 through 0** contain the values 0 or 1. For the output tab, they are checkmarks for output channels allowing you to select 0 (unchecked) or 1 (checked).

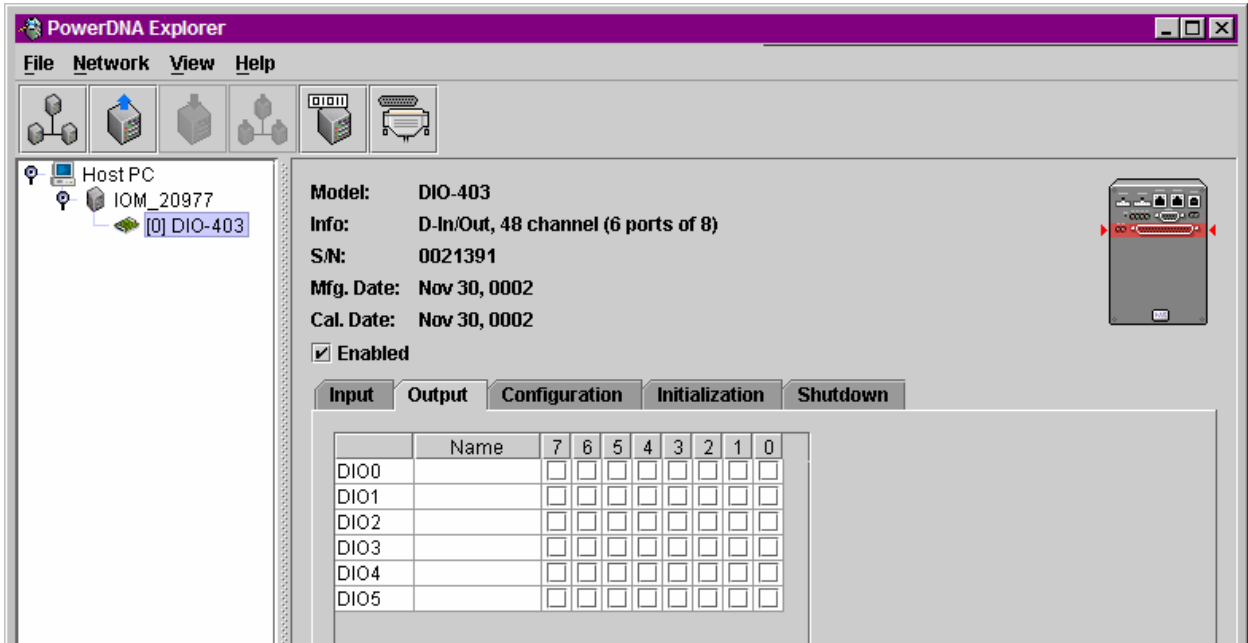


Figure 3-17. Example of DIO-403 Layer Outputs

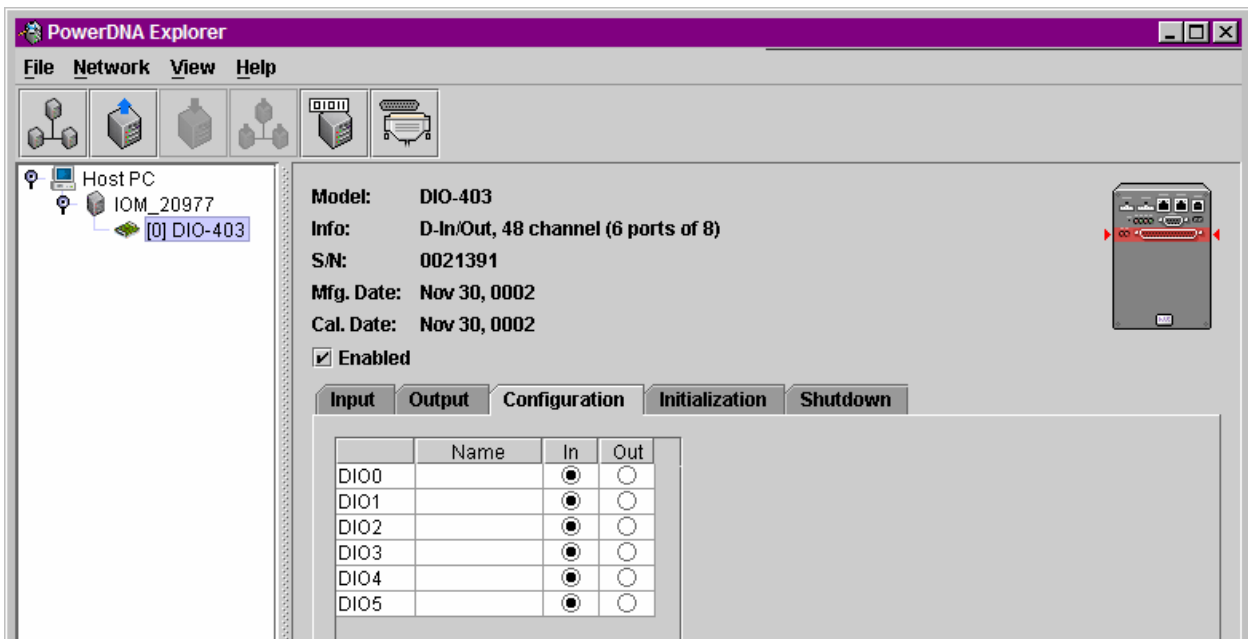


Figure 3-18. Example DIO-403 Layer Configuration

Configuration tab gets/sets the current input/output directions per port. It contains the following columns:

- The unnamed first column contains the channels.
- **Name** is a user-defined string.
- **In/Out** contains toggle switches to select whether the channel is to be used for input or for output.

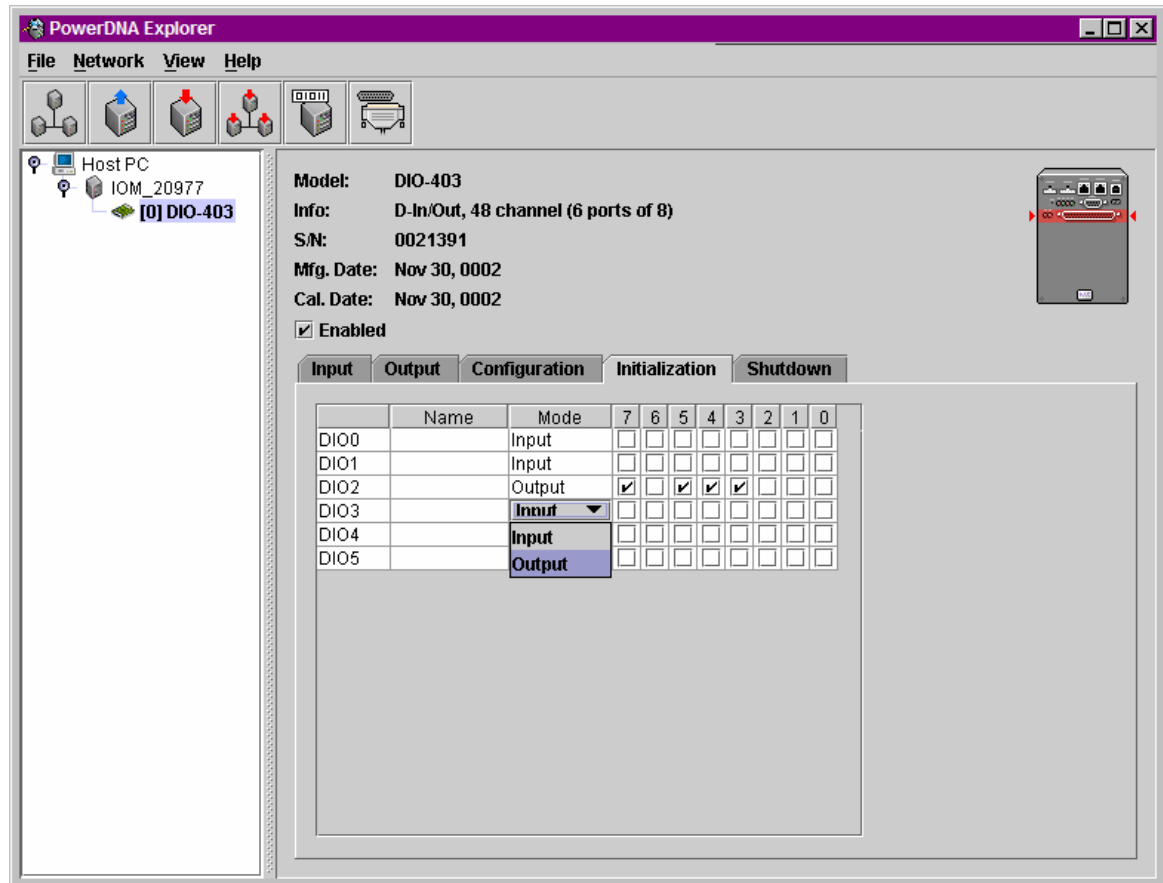


Figure 3-19. Example DIO-403 Layer Initialization

Initialization/Shutdown tabs allow you to set a port as input or output, and set output values. They contain the following columns:

- The unnamed first column contains the channel names.
- **Name** is a user-defined string.
- **Mode** specifies whether the channel is input or output.
- **7 through 0** contain the values 0 or 1. They are checkmarks for output channels that allow you to select 0 (unchecked) or 1 (checked).

3.3 Analog Output Layer Settings

We'll use the AO-302 as an example.

NOTE: Use Network >> Read Input Data to see immediate input values in Input tabs. Use Network >> Store Config to save values to the layer.

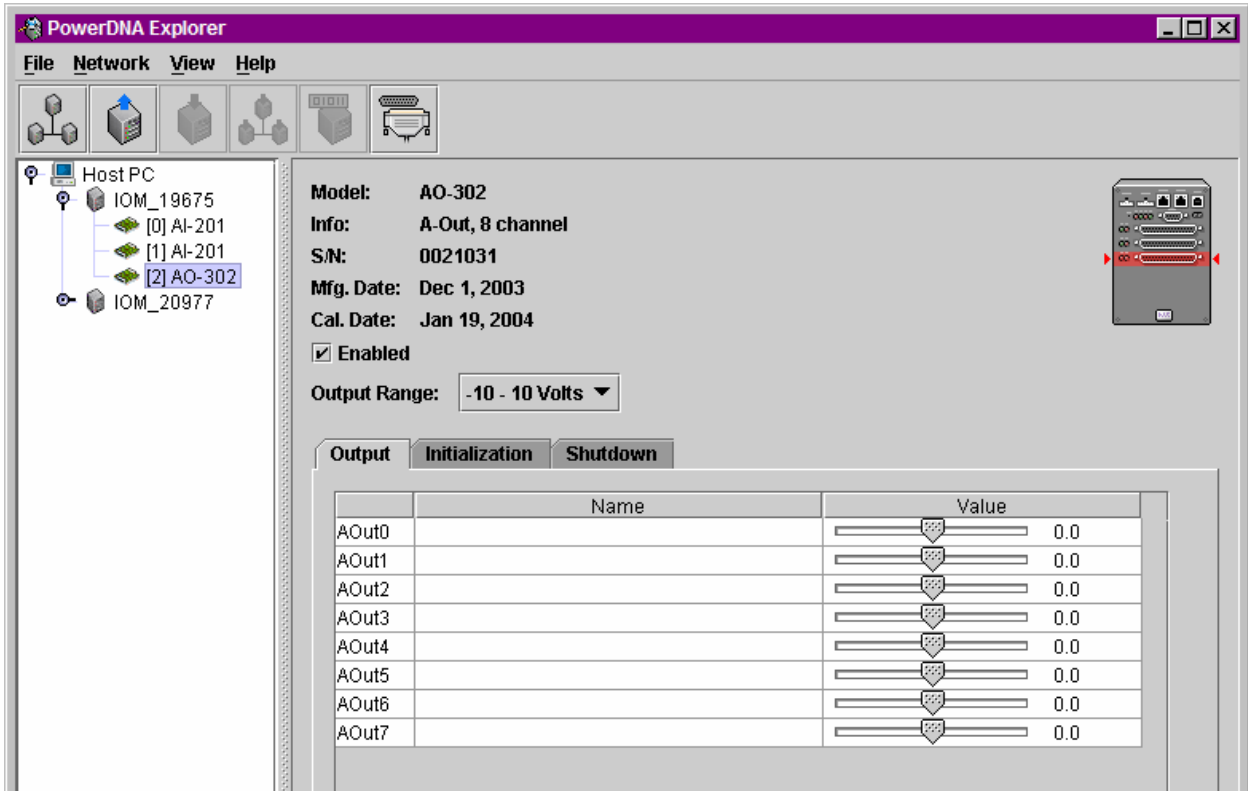


Figure 3-20. Example AO-302 layer

You can change output, initialization, and shutdown values. You can also change Output Range using the combo box, and this only affects values displayed in initialization and shutdown tabs. You can then choose Network >> Store Config to apply all changes to the layer.

Output Range is a popup allowing you to choose between -10...0V, 0...+10V, and -10...+10V.

Output/Initialization/Shutdown tabs switch between settings for init and shutdown states, as well as operation mode configuration.

The **Output, Initialization** and **Shutdown** tabs contain the channel list table, which has the following columns:

- The unnamed first column contains the channel names.
- **Name** is a user-defined string.
- **Value** contains a slider to set the voltage to output from the channel and the numerical voltage value, which you can input directly. The actual voltage depends on the selected output range.

3.4 Analog Input Layer Settings

We'll use the AI-201 as an example to start with. The AI-202 and AI-205 are similar.

NOTE: Use Network → Read Input Data to see immediate input values in Input tabs. Use Network → Store Config to save values to the layer.

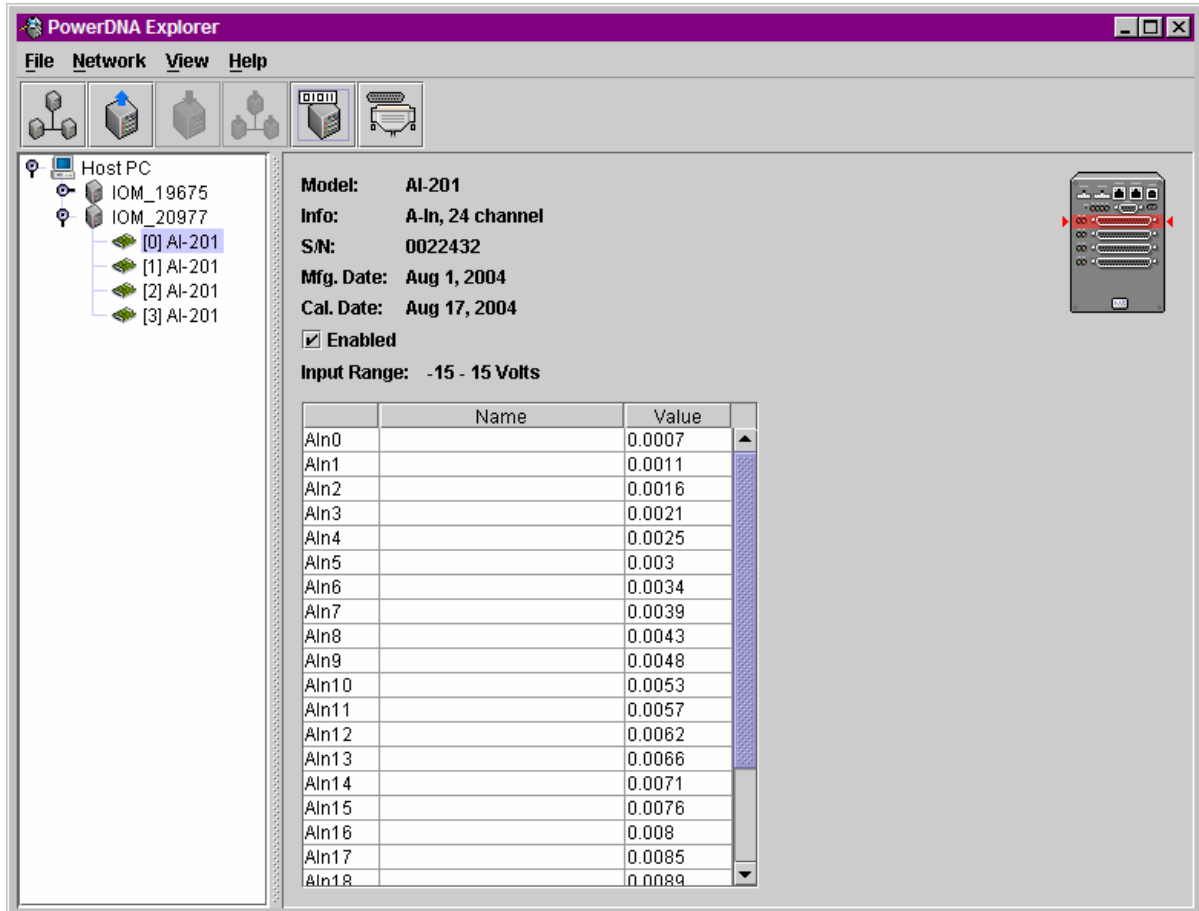


Figure 3-21. Example AI-201 layer

Input Range shows the specified input range. It cannot be changed, and thus is informational only.

The Data table contains the values currently coming into the device. The table is initially blank until you invoke Refresh Data, unless auto-refresh is activated in the preferences dialog. The table has three columns:

- The unnamed first column contains the channel names.
- **Name** is a user-defined string.
- **Value** shows the current value.

3.5 Counter/ Timer Layer Settings

We'll use the CT-601 as an example.

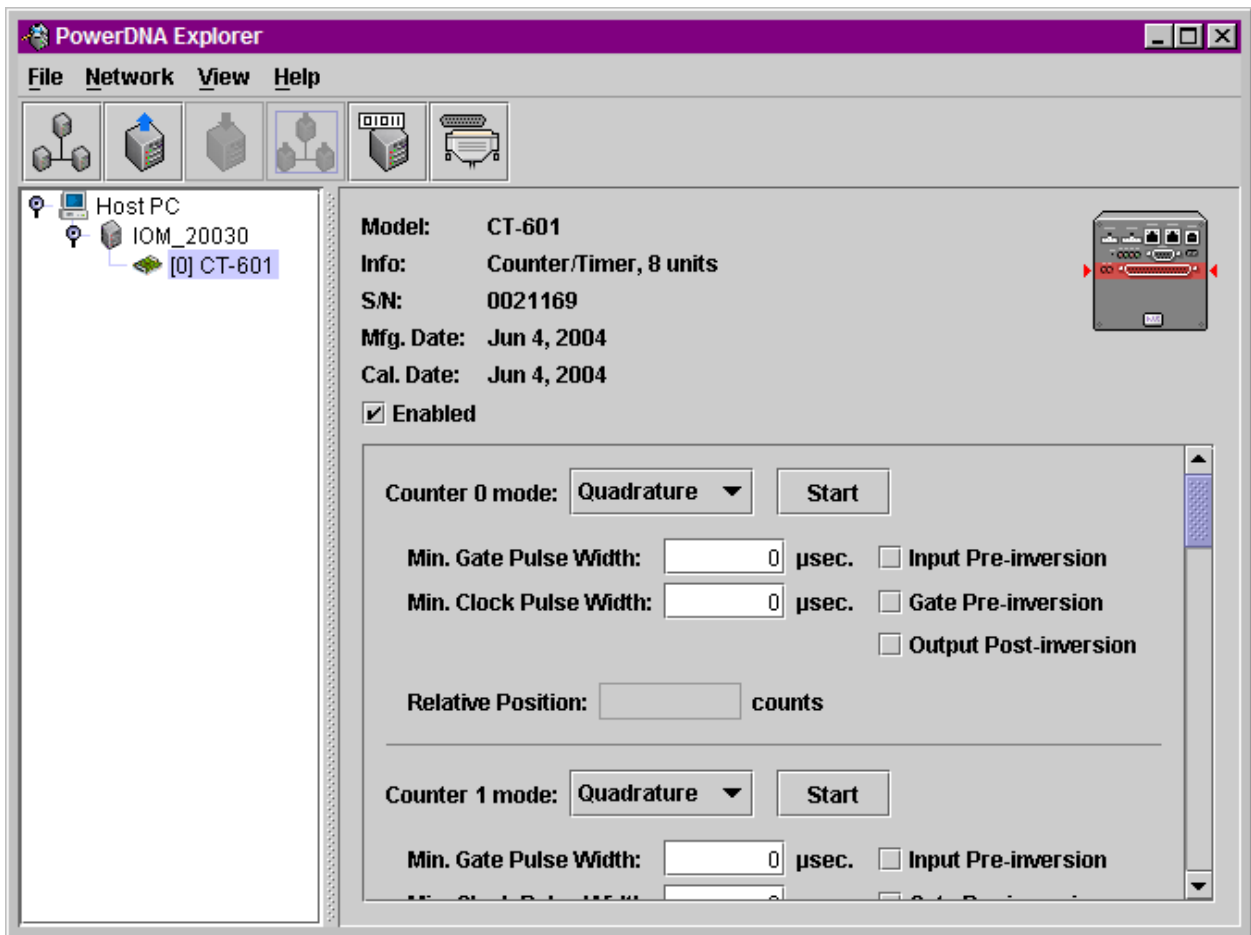


Figure 3-22 Example CT-601 layer

The CT-601 layer has 8 counters. Each counter can be set to one of four different modes: Quadrature, Bin Counter, Pulse Width Modulation (PWM), or Pulse Period. When you change the mode of a counter using the mode combo box, the controls for that counter will change to those appropriate for the mode.

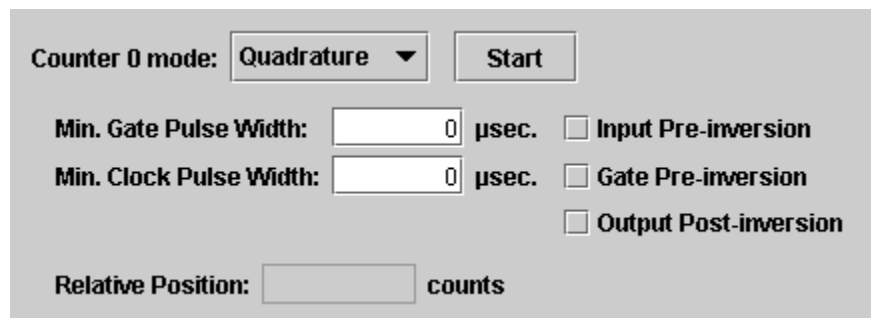


Figure 3-23. Example Quadrature controls

Figure 3-24. Example Bin Counter controls

Figure 3-25. Example Pulse Width Modulation (PWM) controls

Figure 3-26. Example Pulse Period controls

After setting the configuration for a counter, you can choose Network → Store Config to store the settings on the device. Clicking the Start button will also write you configuration to the layer.

Clicking the Start button for a counter will start that counter on the layer. The Start button will turn into a Stop button, and the other controls for that counter will become disabled until you click Stop. While the layer is running, you can choose Network → Read Input Data to retrieve runtime values from the counter, which will display in the read-only text field(s) of the counter control panel.

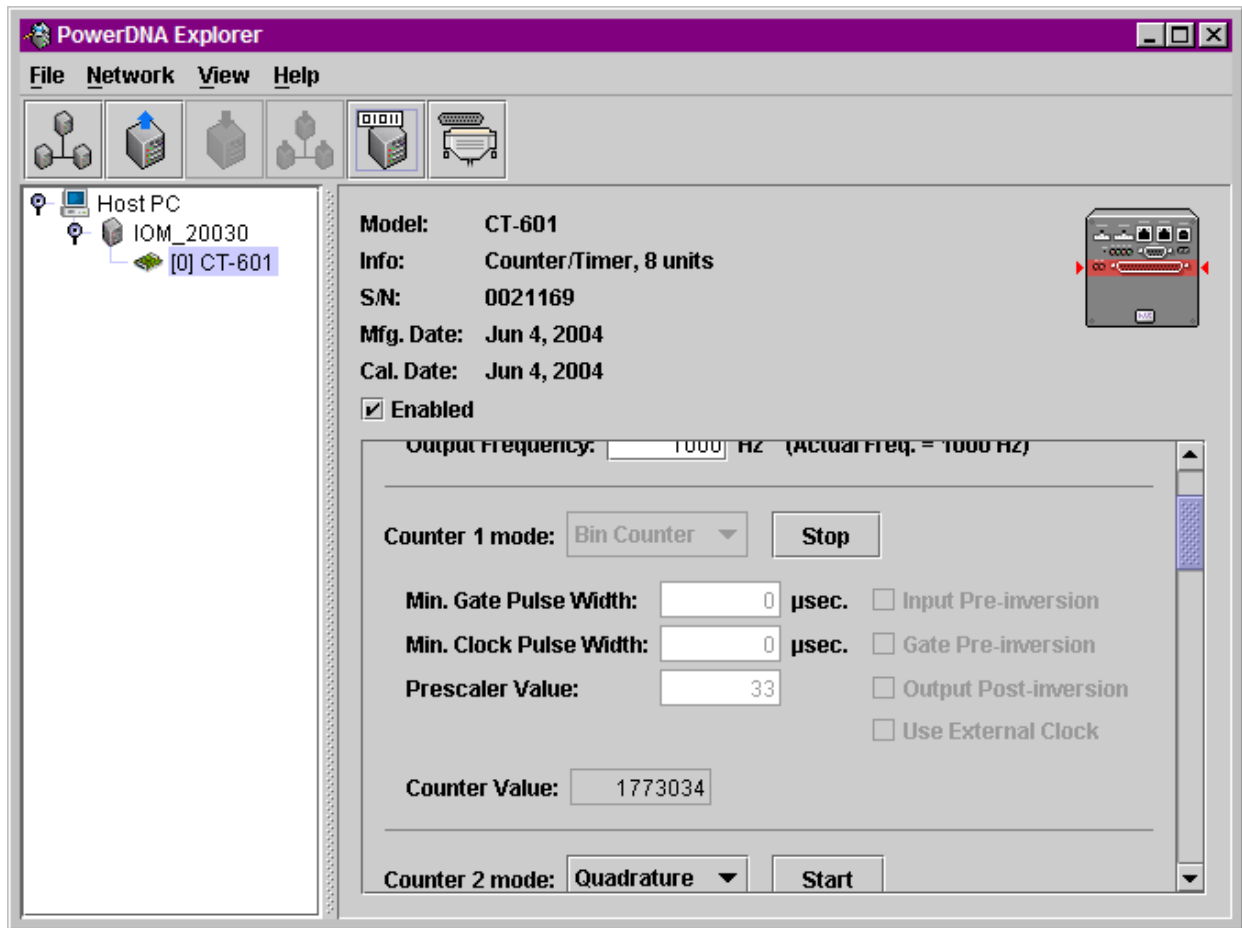


Figure 3-27. Example of Started Counter

Chapter 4 The PowerDNA Core Module

The top two slots of any 5- or 8-slot cube are occupied by the Core Module.

The Core Module consists of a CPU and peripheral devices (RS-232, NIC, SD, etc). The NIC is either a copper (100BaseT) or a Fiber-optic (10/100Base-FX) interface. The CPU is either FreeScale ColdFire (DNA-CM) or PowerPC CPU (DNA-PPC). In addition, an RS-232 port is provided for configuration, and activity lights for status.



Figure 4-1. PowerDNA Core Module (CPU and NIC)

This chapter focuses on the device architecture of the Core Module — no layers.

4.1 Device Architecture of DNA-CM

The CM controller architecture can be represented as follows:

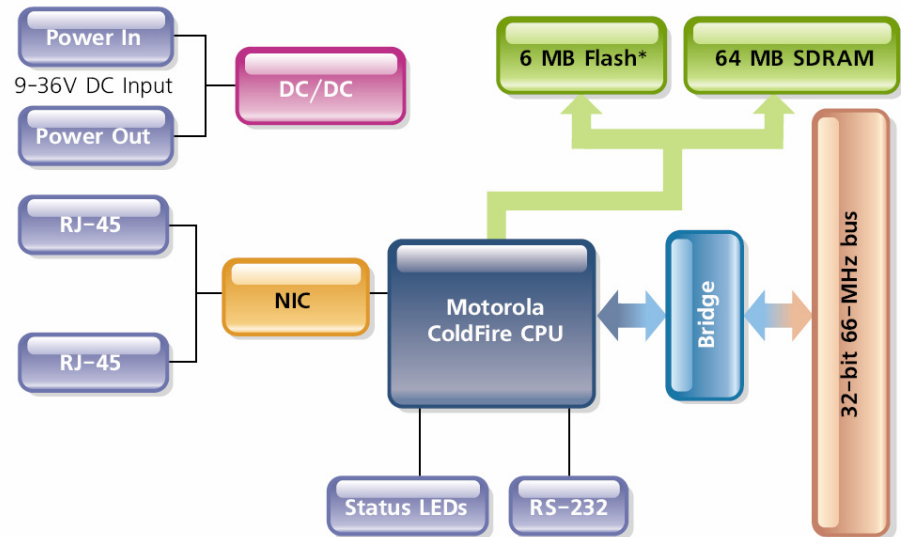


Figure 4-2. FreeScale ColdFire Controller Architecture

The core of the system is a FreeScale (formerly Motorola) ColdFire MCF5272 processor. The processor is directly connected to the following components:

- Network interface MII port
- RS-232 port
- IrDA port
- 2MB user flash memory
- 4MB system flash memory
- 64MB of SDRAM
- Bus bridge
- Control logic
- LEDs
- Watchdog timer with real-time clock (battery backed)

Not all components are available for control from the CPU. The CPU can program flash memory, set the LEDs, set up the watchdog timer, set the real-time clock and use 256 bytes of backed-up memory in the watchdog timer chip. All functions are available at the firmware level only (described in `iom.c/iom.h`).

4.1.1 Device Architecture of DNA-PPC

The PowerPC controller architecture can be represented as follows:

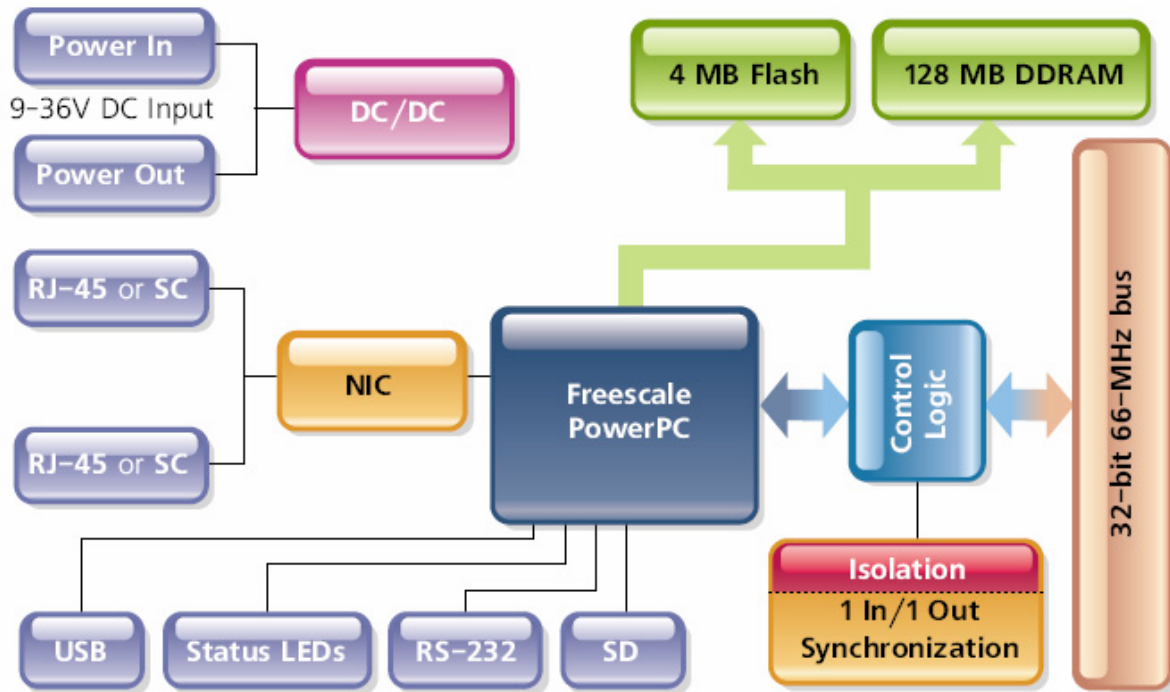


Figure 4-3. PowerPC Controller Architecture

The core of the system is a FreeScale PowerPC MPC5200 400MHz processor. The processor controls the following components:

- Network interface MII port
- RS-232 port
- SYNC port
- 4MB system flash memory
- 128MB of 266MHz DDRAM
- Bus bridge

All functions are available at the firmware level only (iom.c/iom.h).

Chapter 5 Programming Layer-specific Functions

5.1 Overview This chapter describes tools and facilities used for programming module-specific functions — memory maps for various CPUs, register descriptions, procedures for startup, setting parameters, loading/updating firmware, setting up triggers, synchronization, and clock lines.

5.2 Memory Map The ColdFire-based CM has the following memory map:

Device	Start Address	End Address	Size	Description
SDRAM	0x0	0x4000000	64MB	SDRAM_ADDRESS
Interrupt table	0x0	0x400	1024	Processor address map
Firmware load address	0x20000			End address and size depends on firmware size
Firmware start address	0x20400			First execution instruction of firmware
IMM	0x10000000			Memory map register - IMM_ADDRESS
On-board PLD	0x60000000	0x61000000	1MB	EXT_SRAM_ADDRESS
Watchdog timer	0x60008000			IOM_WDTIMER – within PLD access space
Processor RAMBAR	0x80000000			
User flash memory	0x90000000	0x90400000	4MB	FLASHAUX_ADDRESS
Layer – CS2	0xA0000000	0xA00FFFFC	1MB	EXT_DEV_ADDRESS2
Layer – CS3	0xA0100000	0xAFFFFFCC	256M	EXT_DEV_ADDRESS3

The PowerPC-based CM has the following memory map:

Device	Start Address	End Address	Size	Description
SDRAM	0x0	0x8000000	128MB	SDRAM_ADDRESS
Exception table	0x0	0x3000	12k	Processor address map
IMM	0x10000000			Memory map register - IMM_ADDRESS
On-board logic	0xA00E0000	0xA00EFFFFC	64kB	EXT_SRAM_ADDRESS
Watchdog timer	0xA00E8000			IOM_WDTIMER – within PLD access space
Processor RAMBAR	0x80000000			
Layer – CS2	0xA0000000	0xA00FFFFC	1MB	EXT_DEV_ADDRESS2
Layer – CS3	0xA0100000	0xAFFFFFCC	256M	EXT_DEV_ADDRESS3

Device	Start Address	End Address	Size	Description
SDRAM	0x0	0x8000000	128MB	SDRAM_ADDRESS
Flash (parameters)	0xFFC00000	0xFFC0FFFF	64kB	Parameters (64 sectors)
Flash (firmware)	0xFFC10000	0xFFEFFFFF	3MB	Firmware (3MB – 64kB)
Flash (U-Boot)	0xFFF00000	0xFFFFFFFF	1MB	U-Boot

Two address ranges are interesting for host software:

Layer Address Space (0xA0000000 – 0xA00FFFFC and 0xA0100000 – 0xAFFFFFFC). The first address range is dedicated to devices located on CS2 line and accommodates sixteen layers with 64k memory map each. The second address range is designated for fast devices located in CS3 line and it accommodates fifteen devices with 16MB memory map each.

5.2.1 Startup sequence (DNA-CM-5/8)

After reset, the processor starts monitor execution from flash memory. The monitor initializes the processor and the address map, retrieves information from the parameter sector of the flash memory, and tests system memory and other system resources.

If “fwgo” parameter is set to “autorun”, the monitor waits for three seconds for the user to send Ctrl-A (which is transmitted over the serial interface.) If sent, the monitor aborts loading firmware into memory and brings up the monitor command prompt (to load a new firmware version, for example).

Otherwise, the monitor reads the firmware from the flash memory and stores it in RAM. Then, the monitor executes the firmware.

The following parameters are critical to copy firmware and start it from the proper address:

```
fwad: 0xFFE40000
fwgo: 0x1
fwsz: 0x100000
fwcp: 0x20000
fwst: 0x20400
```

These parameters can be reviewed using the “show” command while at the monitor “#>” prompt.

“fwad” is the initial address where firmware is stored. This address shall be set before storing firmware or executing it.

“fwgo” defines whether the monitor should load firmware (1) or should display a command prompt.

“fwsz” defines the size of the stored firmware. Default value is 0x100000 – one megabyte.

“fwcp” defines the address to which the monitor copies firmware from flash memory. The default is 0x20000. Firmware is compiled to run from this address.

“fwst” defines firmware entry point. Firmware entry point follows vector table and is located with offset 0x400 from the beginning of the firmware code.

These parameters are pre-programmed at the factory and there is no known reason for a user to change them.

The monitor command “fwjmp” causes the monitor to load and execute firmware.

5.2.2 Startup Sequence (DNA-PPC-5/8)

After reset, the processor reads the boot-up sequence located at 0xffff100. This command sequence is a part of U-Boot code. U-Boot initializes all major subsystems of the CM including DDRAM and Ethernet interface.

After initializing, U-Boot performs a command list stored in its environment sector under the bootcmd entry. Standard commands to launch firmware are either fwjmp or go 0xffc10000, depending on the version of U-Boot installed. U-Boot then gives up control to the firmware code located at 0xffc10000. Firmware self-expands into the DDRAM, initializes exception table, and starts execution.

5.2.3 Interfacing to the CM Module Using a Serial Interface

There are two ways to set up CM parameters. The first one is the use of serial interface and the second one is the use of DaqBIOS calls.

To connect to the serial interface, the user should connect an extender 9-wire serial cable to the PowerDNA cube (plug male connector) and your PC COM1 serial port (plug female connector). Some cables have a female-to-female connector. If so, you should use a gender-changer.

Set up your terminal to the proper serial port, 57600 bit rate, no parity, eight data bits, and one stop bit.

Alternately, using Start →Run...on the Microsoft Windows desktop, type

\\Program Files\UE\PowerDNA\Firmware\mttty.exe. Click File →Connect.

Once a connection to the PowerDNA cube is established, tap “Enter” once. The PowerDNA cube should respond with either a “DQ>” prompt (this is firmware prompt) or a “#>” prompt (monitor prompt).

Once you see the “DQ>” prompt, you can type “help<enter>” to receive the list of all available commands.

The following commands are available:

```
DQ> help
```

```
help Display this help message          help
set Set parameter                       set option value
show Show parameters                    show
store Store parameters (flash)          store
mw Write wr <addr> <val> (hex)         mw
mr Read rd <addr> (hex)                mr
time Show/Set time                     time [mm/dd/yyyy] [hh:mm:ss]
pswd Set password                       pswd {user|su}
ps Show process state #                 ps [value]
test Test something                     test [test number]
simod System Init/Module Cal            simod [routine]
reset Reset system                      reset [all]
dqping Send DQ_ECHO to <mac addr>      dqping [MAC|IP]
mode Set current mode                   mode
{init|config|oper|shutdown} [ID]
log Display log content                 log [start [end]] -1 = clear
```

```
ver Show firmware version          ver
devtbl Show all devices/layers     devtbl
netstat Show network statistics    netstat
```

One of the most useful commands is “show”:

```
DQ> show
```

```
name: "IOM_22811"
model: 0x1005
serial: 0022811
mac: 00:0C:94:00:59:1B
fwct: 1.2.0.0
srv: 192.168.0.229
ip: 192.168.0.67
gateway: 192.168.0.1
netmask: 255.255.255.0
udp: 6334
```

This command displays current values of every major PowerDNA cube parameter.

To change parameters, use “set” command (type set for “set” command syntax).

```
DQ> set
```

Valid 'set' options:

```
name: <Device name>
model: <Model id>
serial: <Serial #>
mac: <my ethernet address>
fwct: <autorun.runtype.portnum.umports>
srv: <Host IP address>
ip: <IOM IP address>
gateway: <gateway IP address>
netmask: <netmask IP address>
udp: <udp port (dec)>
```

For example, to set a new IP address, type:

```
DQ> set ip 192.168.100.100
```

Other parameters can be changed the same way. Once parameters are set, however, you have to store them into non-volatile flash memory:

```
DQ> store
```

```
Flash: 1212 bytes of 1212 stored! CRC=0x8975E34A Old=0x8975E34A
Configuration stored
```

```
DQ>
```

After parameters are stored, the you should reset the firmware (start firmware execution from the beginning without full hardware reset):

```
DQ> reset
Stopping...
```

```
DaqBIOS (C) UEI, 2001-2004. Running PowerDNA Firmware
Built on 16:39:15 Oct 1 2004
Initialize uC/OS-II (Real-Time Kernel v.252)
Configuration recalled
3 device detected
```

Address	Irq	Model	Option	Phy/Virt	S/N	Pri	DevN
0xA0000000	2	205	1	phys	0023115	10	0
0xA0010000	2	205	1	phys	0023117	20	1
0xA0020000	2	205	1	phys	0023119	30	2

```
-----
Current time: 18:53:45 11/01/2004
IOM: TCP/IP/DQ stack. MAC=00:0C:94:00:59:1B
```

To perform a full hardware reset, use:

```
DQ> reset all
```

The full reset performs a physical reset of the CPU and initiates the whole startup sequence.

Some commands (`mr`, `mw`, `set`, and `store` particularly) require entering a user password. Once the password is entered, these commands become enabled until firmware reset. There are two levels of password protection available. The first is user level and the second is super-user level. Super-user level is currently used only for updating firmware over the Ethernet link.

`DQ> pswd user` sets up a user level password. First, you'll be asked about your old password and then (if it matches) to enter the new password twice.

`DQ> pswd su` sets up a super-user level password. First, you'll be asked about your old super-user password and then (if it matches) to enter the new super-user password twice.

PowerDNA cubes come with the default password set to "powerdna".

Some DaqBIOS commands require clearing up user or super-user passwords. Use `DqCmdSetPassword()` before calling these functions. The PowerDNA API Reference Manual notes which functions are password-protected.

Another useful command is "`devtbl`". This command displays all I/O layers found and initialized by firmware along with assigned device numbers.

Use these device numbers in host software to address these devices.

Priority tells in which order device drivers are located in the device stack. A device with a lower priority number receives a shared interrupt first. The firmware sets up device driver priorities when it registers device drivers.

"`simod`" is a command for system initialization and module calibration.

"`simod 0`" is used to initialize initial layer parameters – serial number, option, etc. We do not recommend use of this command in the field.

“simod 1” allows layer calibration. Different layers have different calibration procedures, explained in respective sections of this document.

“simod 3” allows you to perform factory tests – this is a non-destructive command.



WARNING: Once you use the “simod 0” command, the layer warranty is void.

5.2.4 Setting Parameters

Using the serial interface, you can set up the following parameters:

```
name: <Device name>
model: <Model id>
serial: <Serial #>
  mac: <my ethernet address>
fwct: <autorun.runtype.portnum.umports>
  srv: <Host IP address>
  ip: <IOM IP address>
gateway: <gateway IP address>
netmask: <network mask>
udp: <udp port>
```

“**Name**” sets the device name (up to 32 characters)

“**Model**” sets the device model (factory programmed, do not change). Valid values are 0x1005 – 100-Base-T five-layer PowerDNA cube, 0x1008 – 100-Base-T eight-layer PowerDNA cube, 0x1105 – 100-Base-FX (fiber optics) five-layer PowerDNA cube, 0x1108 – 100-Base-FX eight-layer PowerDNA cube.

“**Serial**” sets the PowerDNA cube’s serial number (factory programmed, do not change)

“**MAC**” sets the PowerDNA cube’s MAC Ethernet address (factory programmed, do not change)

“**fwct**” defines the behavior of the monitor upon boot-up. Valid values for “**autorun**” are zero – stay in monitor after initial boot sequence, or one – copy firmware to SDRAM memory location and execute from there. Valid values for “**runtype**” are TYPE_IOM=2,

TYPE_AUTO =4 and TYPE_SA=8. For normal operation, use TYPE_IOM.

“**portnum**” and “**umports**” parameters are reserved in the current release and should be set to zero.

“**Srv**” sets the host IP address. You have to set the host IP address only if raw Ethernet protocol is in use (used in homogenous IOM networks only.) This parameter is ignored when the PowerDNA cube is used over the UDP protocol or from the host.

“**IP**” specifies the IOM IP address. This is the most important parameter a user must change to allow the PowerDNA cube to be visible on the network. The PowerDNA cube responds to every UDP packet containing a DaqBIOS prolog sent to this address. Since the current release does not support DHCP, the user should set up the IP address.

“**gateway**” specifies where the PowerDNA cube should send an IP packet if a requested IP packet exists outside of the PowerDNA cube network (defined by the network mask). Ask your system administrator if you use your PowerDNA cube on the office network.

“**netmask**” specifies what type of subnet the PowerDNA cube is connected to. The factory sets netmask to type C IP network – 254 nodes maximum

“**udp**” specifies what port the firmware should use if a network packet originated from this PowerDNA cube without a previous request from the host side. If the PowerDNA cube replies to a DaqBIOS packet, it uses the source IP address from the IP packet header and source UDP port from UDP packet header.

Let’s assume that user wants to connect a PowerDNA cube to the dedicated network (secondary NIC adapter in the host PC).

Let’s also assume that host IP address on this dedicated network is:

```
IP address: 192.168.100.28
Network mask: 255.255.255.0
Gateway: ignored
DNS: ignored
```

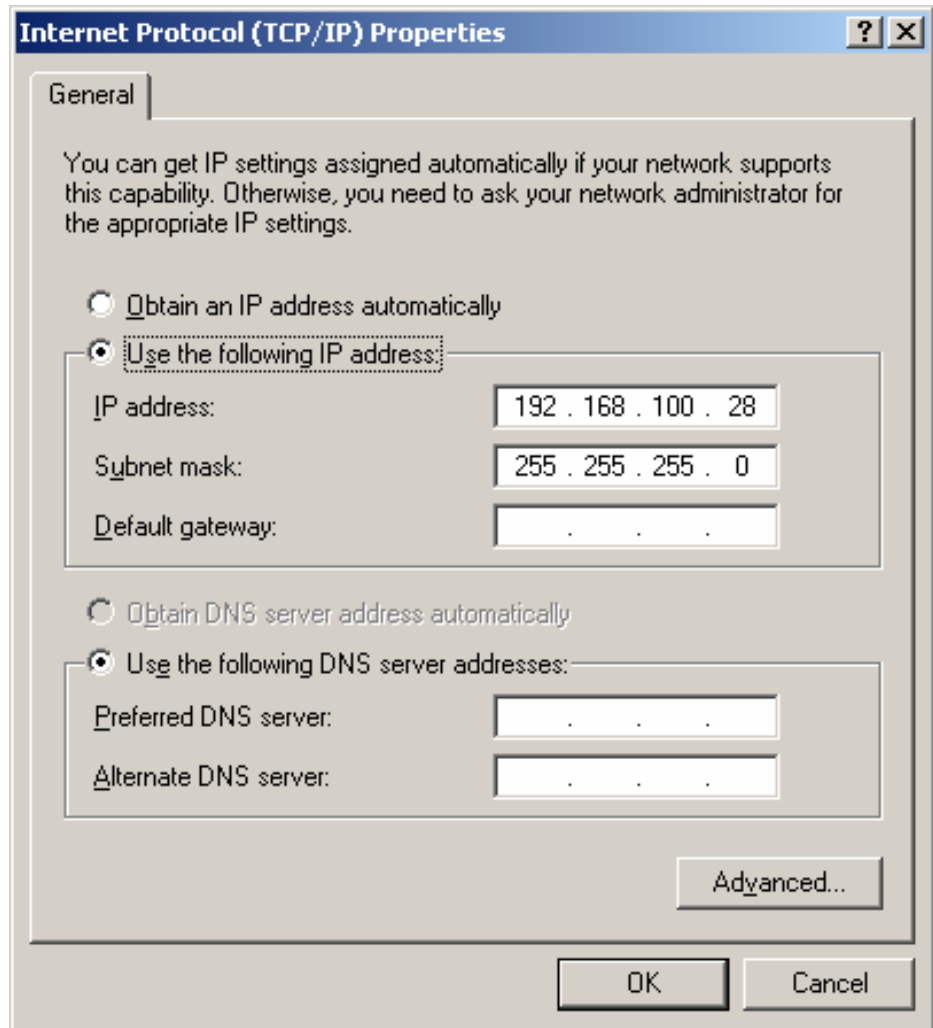


Figure 5-1. Changing the IP Address

Set PowerDNA cube address to any address in the range of 192.168.100.1 through 192.168.0.254 excluding 192.168.100.28 – the host IP address.

For example, type:

```
DQ> set ip 192.168.0.2
```

Then:

```
DQ> store
```

This sequence of commands stores a new IP address in the flash parameter sector. Then, you have to reset the PowerDNA cube.

PowerDNA cubes come from the factory with IP addresses already preset for 192.168.x.x network. The factory IP address can be found on the label located on the back of the PowerDNA cube along with factory MAC address.

After the IP address is set, the user can establish communication with the PowerDNA cube using the PowerDNA Explorer.

5.3 How to Update Firmware

See the Appendix for this information.

5.3.1 Clock and Watchdog Access

To show and set up the date and time, use the “time” command, as follows:

```
DQ> time  
Current time: 17:39:22 11/01/2004
```

To set up time of the day, enter:

```
DQ> time 17:40:00
```

To set up date, enter:

```
DQ> time 11/03/2004
```

Date and time are stored in the battery-backed real-time clock chip.

5.4 Common Layer Interface

The Common Layer Interface is the protocol used in a PowerDNA cube for communication between the IOM and its layers.

5.4.1 Channel List

A channel list specifies what channels and in which sequence each should be acquired/output. Every layer has its own specific set of channel list flags. The firmware takes care of this hardware dependency. Please see the specific layer description to find out what channel list flags are supported.

Users should use the following flags, generalized for all layers.

```
// Channel list entries definition - lower 16 bits are reserved for  
channel number
```

```
// gain and special, module-specific settings
#define DQ_LNCL_NEXT      (1UL<<31)    // channel list has next entry
#define DQ_LNCL_INOUT    (1UL<<30)    // input or output subsystem
#define DQ_LNCL_SS1      (1UL<<29)    // subsystem (high)
#define DQ_LNCL_SS0      (1UL<<28)    // subsystem (low)
#define DQ_LNCL_IRQ      (1UL<<27)    // fire IRQ
#define DQ_LNCL_NOWAIT   (1UL<<26)    // execute this step but don't
// wait
// for the next CV
#define DQ_LNCL_SKIP     (1UL<<25)    // execute this step and discard
// data
// for the next CV
#define DQ_LNCL_CLK      (1UL<<24)    // wait for the next channel list
// clock
#define DQ_LNCL_CTR      (1UL<<23)    // clock counter once
#define DQ_LNCL_WRITE    (1UL<<22)    // write to the channel but not
// update
#define DQ_LNCL_UPDALL   (1UL<<21)    // update all written channels
#define DQ_LNCL_TSRQ     (1UL<<20)    // copy TS along with data (i+=2)
#define DQ_LNCL_SLOW     (1UL<<19)    // slow down operation
#define DQ_LNCL_RSVD2    (1UL<<18)    // reserved
#define DQ_LNCL_RSVD1    (1UL<<17)    // reserved
#define DQ_LNCL_RSVD0    (1UL<<16)    // reserved
#define DQ_LNCL_DIFF     (1UL<<15)    // differential mode
```

There are a few helper macros defined to simplify setting gain and subsystem flags.

```
#define DQ_LNCL_GAIN(G) ((G & 0xf)<<8) // set gain

#define DQ_LNCL_GETGAIN(E) ((E & 0xf00)>>8) // pull out gain
#define DQ_LNCL_GETCHAN(E) (E & 0xff) // pull out channel
#define DQ_EXTRACT_SS(flags) (((flags) & (LNCL_SS1 | LNCL_SS0))>> 28)
#define DQ_EXTRACT_DIR(flags) (((flags) & LNCL_INOUT) >> 30)
#define DQ_SS_DIR(ss, dir) (((ss) << 1) | (dir))
```

The configuration flags serve various functions:

DQ_LNCL_NEXT - specifies that there is a following channel list entry in the channel list. A channel list entry without this flag set is considered the last one. Advanced and ACB functions add this flag automatically

DQ_LNCL_INOUT - specifies whether this is an input or output channel for multifunction layers

DQ_LNCL_SS1 - specifies the subsystem to which the channel belongs. Do not use for single-subsystem layers

DQ_LNCL_SS0 - specifies the subsystem to which the channel belongs. Do not use for single-subsystem layers

DQ_LNCL_IRQ - causes the layer to fire an IRQ upon processing this entry. Required for special real-time cases

DQ_LNCL_NOWAIT – causes the layer to temporarily “forget” about the CV clock and start execution of the next channel list entry right after the current one is completed

DQ_LNCL_SKIP – prohibits storing the data specified in this channel list entry into the data output FIFO or prohibits advancing the data input FIFO pointer. This flag is used to increase the settling time

DQ_LNCL_CLK – causes the channel list machine to wait for the next channel list clock. Normally, the state machine executes the whole channel list on a single CL clock.

DQ_LNCL_CTR – perform a pulse on the selected line. This flag is used for synchronization purposes

DQ_LNCL_WRITE – write the output to the double-register but do not propagate the physical signal to the output.

DQ_LNCL_UPDALL – clock all output channel double-registers to update them simultaneously. This entry is usually used with the DQ_LNCL_WRITE entry when the user needs to write data to the output channels sequentially and update them at the same time. In this situation, the user should use the DQ_LNCL_WRITE flag for every entry. To update all outputs with previously written values, the DQ_LNCL_WRITE flag should be combined with the DQ_LNCL_UPDALL flag.

DQ_LNCL_TSRQ – insert a timestamp into the output data

DQ_LNCL_SLOW – double the settling time for this channel

DQ_LNCL_DIFF – acquire the channel in differential mode (rather than single-ended or pseudo-differential)

The channel number occupies the first eight bits of the channel list entry. The maximum number of channels on one device cannot be larger than 256.

Bits [11..8] contain gain information. The number of gains and the gain are specific for every layer type. See powerdna.h for layer specific gain macros.

5.4.2 Configuration Flags

Configuration flags occupy a 32-bit configuration word. The upper part of the configuration word contains layer-specific flags.

```
// Standard part (lower 16 bits) of layer configuration word
// Please notice that for multiple-subsystem layers one should pass
// multiple configuration uint32s in config_io()
//
#define DQ_LN_TSCOPY      (1L<<18) // copy timestamp along with the
// data
#define DQ_LN_MAPPED     (1L<<15) // For WRRD (DMAP) devices
#define DQ_LN_STREAMING  (1L<<14) // For RDFIFO devices - stream the
//FIFO data automatically
// For WRFIFO - do NOT send reply
// to WRFIFO unless needed
#define DQ_LN_RECYCLE    (1L<<13) // if there is no data taken/
// available
// overwrite/reuse data
```

```
#define DQ_LN_GETRAW      (1L<<12) // force layer to return raw
// unconverted data
#define DQ_LN_TMREN      (1L<<11) // enable layer periodic timer
#define DQ_LN_IRQEN      (1L<<10) // enable layer irqs
#define DQ_LN_PTRIGEDGE1 (1L<<9)  // stop trigger edge MSB
#define DQ_LN_PTRIGEDGE0 (1L<<8)  // stop trigger edge: 00 -
// software, 01 - rising,
// 02 - falling
#define DQ_LN_STRIGEDGE1 (1L<<7)  // start trigger edge MSB
#define DQ_LN_STRIGEDGE0 (1L<<6)  // start trigger edge: 00 -
// software,
// 01 - rising, 02 - falling
#define DQ_LN_CVCKSRC1   (1L<<5)  // CV clock source MSB
#define DQ_LN_CVCKSRC0   (1L<<4)  // CV clock source 01 - SW, 10 -
// HW, 11 -EXT
#define DQ_LN_CLCKSRC1   (1L<<3)  // CL clock source MSB
#define DQ_LN_CLCKSRC0   (1L<<2)  // CL clock source 01 - SW, 10 -
//HW, 11 -EXT
#define DQ_LN_ACTIVE     (1L<<1)  // "STS" LED status
#define DQ_LN_ENABLED    (1L<<0)  // enable operations
```

`DQ_LN_ACTIVE` is needed to switch on the "STS" LED on CPU layer.

`DQ_LN_ENABLE` enables all operations within the layer

`DQ_LN_CLCKSRC0` selects the internal channel list clock (CL) source as a time base. AI-201 supports the CL clock only where the time between consecutive channel readings is calculated by the rule of maximizing setup time per channel. If you'd like to clock the CL clock from an external clock source such as SYNCx line, set the `DQ_LN_CLCKSRC1` flag as well.

`DQ_LN_CVCKSRC0` selects the internal conversion clock (CV) source as a time base. Setting CV clock allows having an equal time period between conversions of different channels. It is mostly used when the user is interested in a phase shift between different channels.

The user can select either the CL or CV clock as a time base. If both clocks are selected, the CL clock is taken as a time base and the CV clock determines the delay between converting channels (i.e. setting time.)

IS (Isolated Side)

`DQ_LN_STRIGEDGE0`, `DQ_LN_STRIGEDGE1` define the start trigger edge and source. The source can be either software command or external trigger edge.

`DQ_LN_PTRIGEDGE0`, `DQ_LN_PTRIGEDGE1` define the stop trigger edge and source. The source can be either software command or external trigger edge.

`DQ_LN_TSCOPY` – copy timestamp at the end of every channel list

`DQ_LN_MAPPED` - set this flag to declare DMap mode

`DQ_LN_STREAMING` - set this flag to declare ACB mode

`DQ_LN_RECYCLE` - this flag affects output operation. If this flag is set and layer does not receive output data, it will recycle old data until new data is available; otherwise, the layer will stop at the last value output

DQ_LN_GETRAW - tells the layer to return uncalibrated unconverted data. This flag makes sense only for layers with software calibration (AI-225, for example). Moving calibration and conversion of data to host unloads IOM processor

DQ_LN_TMREN - use a real-time timer to retrieve data from the PowerDNA cube. When this mode is selected, the firmware programs the layer to store one channel list worth of data in the buffer. On a timer tick, the firmware transfers this data from the layer output buffer to the packet. This function is used when the hardware allows only a selected set of update rates, but the user needs something in between. For example, AI-225 can convert data with fixed frequency equal $6.875\text{Hz} * 2^n$, where $n = [0\dots9]$. To receive an exactly 500Hz data stream from this layer, one should specify that this layer be updated upon a timer tick.

DQ_LN_IRQEN - use interrupts to retrieve data from the layer output buffer via packets. This is preferable mode of operation.

5.4.3 EEPROM User Area Access

Every I/O layer has an E²PROM chip that contains 2048 bytes of layer-specific information.

Model and option numbers identify every layer. The model number is hard-coded inside layer logic and option numbers are stored inside E²PROM.

E²PROM is divided into certain access areas (some of them can be missing in different layer types):

```
typedef struct {
    DQEECMNDEVS ee;
    DQCALSET_xxx_ calset;
    DQOPMODEPRM_xxx_ opmodeprm;
    DQINITPRM_xxx_ initprm;
    DQSDOWNPRM_xxx_ sdownprm;
    DQCNames_xxx_ cname;
} DEVEEPROM_xxx_, *pDEVEEPROM_xxx_;
```

The first part of the layer E²PROM is common device information defined as:

```
typedef struct {
    /* header is standard for all devices */
    /* superuser access */
    uint16 model; /* device model to verify EEPROM identity */
    uint16 option; /* device option */
    uint16 total; /* total EEPROM size - EEPROM read is expensive
*/
/* if this field <32 or >2048 read all2048 bytes
*/
    uint32 sernum; /* serial number - pad to %07d when printing */
    uint32 mfgdate; /* manufacturing date: 0xmmddyyyy */
    /* user access */
    uint32 caldate; /* calibration date: 0xmmddyyyy */
    uint32 calexp; /* calibration expired: 0xmmddyyyy */

    /* header is followed by device-specific data structures */
} DQEECMNDEVS, *pDQEECMNDEVS;
```

CALSET_xxx_ contains layer calibration information. Firmware writes this information automatically upon entering initialization mode.

OPMODEPRM_xxx_ contains layer parameters for operation mode. For example, AI-201 has the following parameters stored:

```
typedef struct {
    uint32 chlst[AI201_CHAN];    // channel list - full
    uint32 conf;                // control word - layer API flags
    uint32 cvclk;               // CV clock
    uint32 clclk;               // CL clock
    uint32 trig;                // trigger configuration
    ...
} DQOPMODEPRM_201_, *pDQOPMODEPRM_201_;
```

This structure varies from one major firmware revision to another.

When the firmware switches the layer into operation mode, it processes stored configuration information as it would process configuration parameters received from host. All working fields in the internal device information structure are filled and the unit is ready to switch into operation mode. By programming the DQOPMODEPRM structure ahead of time and storing it into E²PROM, you can avoid programming the IOM every time before switching into operation mode.

INITPRM_xxx_ contains initial I/O directions and output levels. The firmware sets up the direction and the level on every output line on entering initialization state.

SDOWNPRM_xxx_ contains final I/O directions and output levels. The firmware sets up the direction and the level on every output line on entering shutdown state.

CNAMES_xxx_ contains channel names. The length of the channel names depends on the layer type. Only 512 bytes are allocated for channel names. Thus, AI-205 layer (four channels) can have channel names as long as 32 characters while DIO-403 channel names (48 channels) cannot be longer than 10 characters.

There is a set of functions written to read, write, and store these parameters into E²PROM. Functions `DqCmdGetParameters()`/`DqCmdSetParameters()` access modal parameters, while `DqCmdSaveParameters()` stores parameters into E²PROM.

5.5.4 PowerDNA Layer Signaling

Setting up triggering, synchronization, and clocking lines

Most PowerDNA layers have the following interconnection diagram:

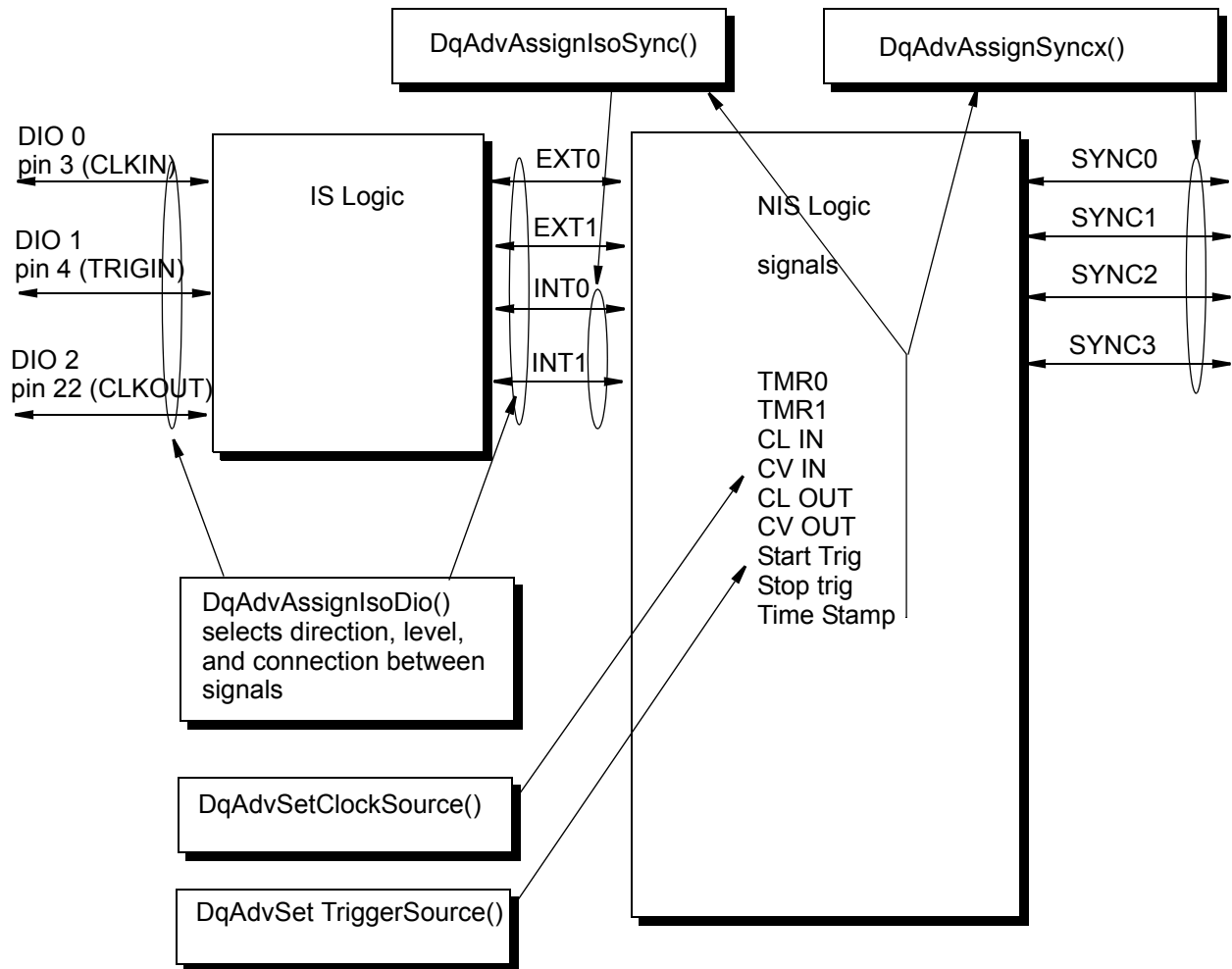


Figure 5-2. CM Interconnection Diagram

- DIO0/CLKIN – pin 3 on the FJIO1 DB-37 connector. By default, this pin is an input, connected to the ISO_EXT0 synchronization line and through this line to the NIS logic
- DIO1/TRIGIN – pin 4 on the FJIO1 DB-37 connector. By default this pin is an input, connected to the ISO_EXT1 synchronization line and through this line to the NIS logic
- DIO2/CLKOUT – pin 22 on the FJIO1 DB-37 connector. By default this pin is an output connected to the ISO_INT0 line from the NIS logic

The PowerDNA API exposes six specially designated functions to control these lines, as follows:

- **DqAdvSetClockSource()**

This function selects external clock source for CL (or CV) clock. The clock can be selected from internal sources, EXTx lines (signals from the isolated side), and SYNCx interface signals (inputs)

- **DqAdvSetTriggerSource()**

This function selects external clock source for start and stop trigger. Clock can be selected from internal sources, EXTx lines (signals from the isolated side), and SYNCx interface signals (inputs)

- **DqAdvAssignIsoDio()**

This function selects direction and signal assignment for external DIO line. EXT0/1 lines are assigned to DIO0/1 lines when DIO lines are in the input state.

- **DqAdvAssignIsoSync()**

This function selects signal assignment for INT lines. This function allows selecting what signal from isolated side of the layer logic will be assigned to INTx lines. Signals can be selected from internal clock sources and SYNCx lines.

- **DqAdvAssignSyncx()**

This function selects a signal for each of the SYNCx lines. When a SYNC line is selected, it switches to the output state. All other layers "listen" to this command on the system bus and release that SYNC line from use (switch to the input mode). This organization prevents two layers from driving the same line.

- **DqAdvWriteSignalRouting()**

This function writes and activates selected signal routing. This function transfers created configuration to the cube and activates it. Cube sends current synchronization configuration as a reply.

NOTE: Please note that to take advantage of using external clocks for the layer clock and/or trigger, the source should be selected as external. This means that, in clocking configurations, the following bits should be set up:

```
DQ_LN_CLKSRC1 - external CL clock is selected  
DQ_LN_STRIGEDGE1 - external start trigger is selected  
DQ_LN_PTRIGEDGE1 - external stop trigger is selected
```

If internal sources are selected for those signals, all external signal configurations do not affect layer clocking.

The same interface applies to the CPU layer. The CPU layer has one external input and one output routable to the SYNCx interface as well as multiple clocks. It is possible to include an IEEE 1588 implementation with an atomic clock (1us) resolution in the future.

5.6 Register Map and Description All CTU registers are located at addresses starting at Base+DQ_CLI_CTUxS, where x is the CTU number 0-7. The I/O FIFO uses the standard PowerDNA FIFO locations starting at Base+0x1800.

Register Offset	Name	Description
DQ_CTU_STR	CTU status register	Counter/timer status register contains status bits related to CTU functionality. (read)
DQ_CTU_CTR	CTU control register	Counter/timer control register contains control bits related to CTU functionality. (write)
DQ_CTU_CCR	CTU counter control register	Counter/timer Counter Control register – defines mode of the operation of the counter and prescaler. (write)
DQ_CTU_PS	Prescaler Divider	Program prescaler divider (PS) (default value – 0) and read back current value of the prescaler counter. (read/write)
DQ_CTU_CR	Count Register	Current value of the count register. (read)
DQ_CTU_LR	Program Load Register	Write access sets new value of the load register LR also copying the same value into the count register CR. (write)
DQ_CTU_IDBC	Clock Debouncing Register	Program input clock-debouncing register IDBC. CTU will expect input clock to remain stable for the specified number of 66MHz clocks before processing/ qualifying it. (write)
DQ_CTU_IDBG	Gate Debouncing Register	Program input gate-debouncing register IDBG. CTU will expect input gate line to remain stable for the specified number of 66MHz clocks before processing/ qualifying it. (write)
DQ_CTU_PC	Period Count Register	Period count register PC is used in a -measurement modes when averaging for the multiple periods is required because of the high-speed or unstable nature of the incoming signal. Results of the measurement are accessible only after specified number of periods on the incoming signal are detected. (read/write)
DQ_CTU_CRH DQ_CTU_CR0	CTU Capture Register High	Read: Provide access to the capture register high. Write: Set value of the compare register 0. (read/write)
DQ_CTU_CRL DQ_CTU_CR1	CTU Capture Register Low	Read: Provide access to the capture register low. Write: Set value of the compare register 0. (read/write)
DQ_CTU_TBR	Time Base Register	TBR defines time-base divider for the time-based capture modes. Bit 31 (MSB) of the TBR. (write)
DQ_CTU_FCNTI	Input FIFO Count Register	CTU Input FIFO – FIFO0 Count. (read)
DQ_CTU_FIRQI	Input FIFO IRQ Level	CTU Input FIFO – FIFO0 IRQ. (write)
DQ_CTU_FDTI	Input FIFO Data Register	CTU Input FIFO – FIFO0 Data In. (write)
DQ_CTU_FCNTO	Output FIFO Count Register	CTU Output FIFO – FIFO1 Count. (read)

Register Offset	Name	Description
DQ_CTU_IRQO	Output FIFO IRQ Level	CTU Output FIFO – FIFO1 IRQ level - reserved. (write)
DQ_CTU_FDTO	Output FIFO Data Register	CTU Output FIFO – FIFO1 Data Out. (read)
DQ_CTU_ISR	Interrupt Status Register	ISR shows current status of the enabled interrupts. (read)
DQ_CTU_IER	Interrupt Enable Register	IER is used to specify specific interrupt conditions should generate an interrupt. (write)
DQ_CTU_ICR	Interrupt Clear Register	ICR allows clearing of “fired” interrupt bits. If interrupt condition persists, interrupt will be fired again. (write)
DQ_CTU_FDDO	Output Data FIFO Register	FDDO is a reserved register used for the time sequencer version of CTU implementation. (write)
DQ_CTU_TEST0	Test Register 0	TEST0 is a reserved test read-only register. In current implementation, read from TEST0 returns 0x01234567. (read)
DQ_CTU_TEST1	Test Register 1	TEST1 is a reserved test read-only register. In current implementation, read from TEST1 returns 0xABCD0123. (read)

The following table shows Counter/Timer Units 0-7 registers with 0x80 offset increment representations.

0x2000-0x207C	DQ_CLI_CTU0S	CTU0 I/O registers
0x2080-0x20FC	DQ_CLI_CTU1S	CTU1 I/O registers
0x2100-0x217C	DQ_CLI_CTU2S	CTU2 I/O registers
0x2180-0x21FC	DQ_CLI_CTU3S	CTU3 I/O registers
0x2200-0x227C	DQ_CLI_CTU4S	CTU4 I/O registers
0x2280-0x22FC	DQ_CLI_CTU5S	CTU5 I/O registers
0x2300-0x237C	DQ_CLI_CTU6S	CTU6 I/O registers
0x2380-0x23FC	DQ_CLI_CTU7S	CTU7 I/O registers

5.7 Register Descriptions

This section lists bit descriptions for various status registers.

0x2000 RD – CT0_STR – CTU0 Status Register

The CTU Status register is used to report current operational status of the counter/timer unit via dedicated bits for every status condition reported. The Status register mirrors some of the ISR (interrupt status register) bits, but it reports current status while ISR reports latched status of the “fired” interrupts

Bit	Name	Description	Reset
31	DQ_STR_EN	When read as 1 indicates that CR is enabled in CT0_CTR DQ_CTR_EN bit.	0
30	DQ_STR_BUSY	When read as 1, indicates that CR is counting or 0 if current counting operation is complete	0
29	DQ_STR_CR0L	When read as 1, indicates that current value of CR < CR0	0
28	DQ_STR_CR0GE	When read as 1, indicates that current value of CR >= CR0	0
27	DQ_STR_CR1	When read as 1, indicates that current value of CR >= CR1	0
26	DQ_STR_IN0	Report current value of direct input pin	0
25	DQ_STR_GT0	Report current value of direct gate pin	0
24	DQ_STR_IN1	Report current value of de-bounced input pin	0
23	DQ_STR_GT1	Report current value of de-bounced gate pin	0
22	DQ_STR_IHL	When read as 1, indicates that 1-0 transition was detected on the input pin since last read from CTx_STR. This bit will be automatically cleared after each read.	0
23	DQ_STR_ILH	When read as 1, indicates that 0-1 transition was detected on the input pin since last read from CTx_STR. This bit will be automatically cleared after each read.	0
22	DQ_STR_GHL	When read as 1, indicates that 1-0 transition was detected on the gate pin since last read from CTx_STR. This bit will be automatically cleared after each read.	0
19	DQ_STR_GLH	When read as 1, indicates that 0-1 transition was detected on the gate pin since last read from CTx_STR. This bit will be automatically cleared after each read.	0
18	DQ_STR_OU	Report current value of output pin	0
17	DQ_STR_IRQ	Read as 1 if interrupt was requested	0
16	DQ_STR_CRH	Report 1 if data is available in CRH	0
15	DQ_STR_CRL	Report 1 if data is available in CRL	0
14	DQ_STR_IFE	Report 1 if input FIFO is empty	0
13	DQ_STR_IFH	Report 1 if input FIFO is at least ½ full	0
12	DQ_STR_IFF	Report 1 if input FIFO is full	0
11	DQ_STR_OFE	Report 1 if output FIFO is empty	0
10	DQ_STR_OFH	Report 1 if output FIFO is at least ½ full	0
9	DQ_STR_OFF	Report 1 if output FIFO is full	0

0x2000 WR – CT0_CTR – CTU0 Control Register

The CTU Control register is used to set and control some parameters of the operation mode of the counter/timer via specific bits and bit fields. Note that the generic interrupt mask/enable/control/status is reported via layer IER (0x1C), IMR(0x20), ISR/ICR (0x24) registers. Layer-specific bits are described later in the section. Status conditions that lead to the interrupt request are enabled/disabled via CTx_CTR register.

The following are the DQ_CT0_CTR bit descriptions for the CTU0 register.

Bit	Name	Description	Reset
31	DQ_CTR_EN	Enable (1)/Disable (0) counter register. When disabled, CR along with pre-scaler and de-bouncer circuitry, freezes its current operation, which may be re-enabled by writing a one to the DQ_CTR_EN bit.	0
30	DQ_CTR_IFE	Input FIFO enable (1) disable(0). Depending on the operation mode, when enabled, fetches one 32-bit word from the input FIFO to the CR0 at the same time the counter register is reloaded with LR value	0
29	DQ_CTR_IFS	Input FIFO transfer size. Used only when DQ_CTR_IFE = 1. 0 - 1 word, 1- 2 words. Defines one (CR0) or two words (CR0/CR1) and is loaded whenever a time "end-of-count" condition is detected	0
29	DQ_CTR_IIE	Enable (1)/Disable (0) inversion of the input pin. Value of the pin is inverted at the input before debouncing circuitry	0
28	DQ_CTR_GIE	Enable (1)/Disable (0) inversion of the gate pin. Value of the pin is inverted at the input before debouncing circuitry	0
27	DQ_CTR_OIE	Enable (1)/Disable (0) inversion of the output pin. When enabled – output pin polarity is inverted at the last stage of creating the output	0
26	DQ_CTR_OU	Current value of the output pin in GPIO mode (valid if DQ_CTR_EN bit = 0 and DQ_CTR_GPIO=1)	0
25	DQ_CTR_OFE	Output FIFO – enable (1)/disable(0). Depending on the operation mode, when enabled, copies one or two 32-bit words from the input CR or CRH/CRL into the output FIFO when counter reaches end of count condition	0
24	DQ_CTR_CLFI	If this bit is set during write to CTR, all input paths will be cleared (CRH/CRL and input FIFO), FIFO will contain 0 samples, and CRH/CRL will be set to 0. Reset input FIFO before initiating any HOST→CTU transfers	
23	DQ_CTR_CLFO	If this bit is set during write to CTR, all output paths will be cleared (CR0, CR1, LR and output FIFO), FIFO will contain 0 samples, and all registers affected will be set to 0. Reset output FIFO before initiating any CTU→HOST transfers	
22	DQ_CTR_CLR	If this bit is set during write to CTR, CTUx will be reset to the default state, and all registers/FIFO will be cleared	

Bit	Name	Description				Reset
21	DQ_CTR_GPIO	If this bit is set, GPIO operation of the "clkout" pin is enabled				0
		DQ_CTR_EN	DQ_CTR_GPIO	DQ_CTR_OU	clkout	
		0	0	x	Remains in a last state	
		0	1	0	0	
		0	1	1	1	
1	x	x	Defined by the current CTU mode			
20-0		Reserved				0

0x2004 WR – CT0_CCR – CTU0 Counter Control Register

The CTU Counter Control register is used to set current mode for the counter and pre-scaler.

The following table lists the CT0_CCR Bit descriptions.

Bit	Name	Description	Reset State
31	DQ_CCR_RE	Enable re-load of the CR by the value loaded in LR when it reaches end of count. End of count is limited by one of the combinations of DQ_CCR_EC2/1/0 bits	0
30-28	DQ_CCR_EC2 DQ_CCR_EC1 DQ_CCR_EC0	Set end of count mode: (0)000 – DQ_EM_CR0, end, when it reaches CR0 (CR=CR0) (1)001 – DQ_EM_CR1, end, when it reaches CR1 (CR=CR1) (2)010 – DQ_EM_FFF, end, when it reaches 0xFFFFFFFF (3)011 – DQ_EM_PC, end, when X periods of the signal are captured. X is defined via CTx_PC (0x2018) register. In width (1/2 period) measurement mode end, when positive part of the input signal is captured. (4)100 – DQ_EM_TBR, end, when the time-base counter reaches 0. Note: All other modes are reserved for future use and will be recognized as a mode 0	0
24-27	DQ_CCR_CRM3 DQ_CCR_CRM2 DQ_CCR_CRM1 DQ_CCR_CRM0	Set counter mode: (0x0)0000 – DQ_CM_CT counter (CR acts as a standard count-up counter, 66MHz base clock used as a PS source) (0x8)1000 – DQ_CM_ECT counter (CR acts as a standard count-upcounter, debounced CLKIN clock used as a PS source) (0x9)1001 – DQ_CM_HP capture ½ period mode (CR captures ½ period of the input signal starting from the rising edge of the de-glitched input and copies it into CRH). (0xA)1010 – DQ_CM_NP capture full period (CR captures length of the full period, copies positive part of the period into CRH and negative (low) into CRL, if CTx_PC > 0 – continue this process increasing CRH/CRL for the length of positive/negative part of every period (0xB)1011 – DQ_CM_QE quadrature encoder mode (0x4)0100 – DQ_CM_TCT same as 0x0 but with trigger (0xC)1100 – DQ_CM_TECT same as 0x8 but with trigger (0xD)1101 – DQ_CM_THP same as 0x9 but with trigger (0xE)1110 – DQ_CM_TNP same as 0x9 but with trigger Note, that all modes, except mode 0 are using debounced CLKIN pin as a clock source for the pre-scaler. Trigger source (Hardware/ Software) is selected using DQ_CCR_TRS bit	0

Bit	Name	Description	Reset State
23	DQ_CCR_PSG	Enable(1)/Disable(0) hardware gate on the prescaler. If enabled, GATE input, when positive, enables pre-scaler counter. Note, that DQ_CTR_EN bit in CTR may be effectively used as a software gate, when DQ_CCR_PSG = 0.	0
22	DQ_CCR_TRS	Select Hardware(1)/Software(0) trigger source for the triggered modes. Hardware-triggered modes will start at low-high transition on the GATE input. In software trigger mode, DQ_CTR_EN bit in CTR should be used as a trigger (DQ_CTR_EN will be cleared at the end of the counting operation if CCR_TRS bit is cleared and triggered mode is selected DQ_CM_Txx)	0
21	DQ_CCR_ENC	This bit complements DQ_CCR_TRS bit and works only in a triggered mode – if set (1), enables auto-clear of the DQ_CTR_EN bit at the end of the current operation.	

5.7.1 Valid EM/CM Combinations for Non-Buffered Modes Refer to the table below for the possible EM/CM combinations (x – valid mode):

	DQ_EM_CR0	DQ_EM_CR1	DQ_EM_FFF	DQ_EM_PC	DQ_EM_TBR	Notes
DQ_CM_CT DQ_CM_ECT DQ_CM_TCT DQ_CM_TECT	x	x	x		x	Use DQ_EM_CR0 for single-clock pulse generation, DQ_EM_CR1 for PWM mode, DQ_EM_FFF for wrap-around counter
DQ_CM_HP DQ_CM_NP DQ_CM_THP DQ_CM_TNP				x		
DQ_CM_QE	x	x	x		x	For continuous non-buffered operation of QE, it is recommended you use DQ_EM_TBR mode, but disable timebase counter by writing 0x1 to TBR

0x2008 WR – CT0_PS – CTU0 Prescaler

Set value of the pre-scaler. Prescaler is a 32-bit count-down counter output of which is used to clock counter register (CR). Source for the prescaler is automatically selected based on current value of the CCR_CRMx bits. Note that if pre-scaler is loaded with 0, it will be by-passed and an input signal will be used as an input clock for the count register CR (but GATE pin if used will still affect the counter).

0x2008 RD – CT0_PS – CTU0 Prescaler Current Value

Read current 32-bit value of the prescaler.

0x200C RD – CT0_LR – CTU0 Load Register

32-bit value, stored in the load register LR, will be loaded into the main counter CR at the beginning of each counting cycle.

0x200C RD – CT0_CR – CTU0 Count Register Current Value

Current value of the count register, latched at the time of the read.

0x2010 WR – CT0_IDBC – CTU0 Input Pin Debouncing Filter Counter Register

Program input clock-debouncing register 32-bit register IDBC. CTU0 will expect input clock to remain stable for the specified number of 66MHz clocks before processing/qualifying it.

0x2014 WR – CT0_IDBG – CTU0 Gate Pin Debouncing Filter Counter Register

Program input gate-debouncing register IDBG. CTU0 will expect input gate line to remain stable for the specified number of 66MHz clocks before processing qualifying it.

0x2018 RD – CT0_PC – CTU0 Current Value of the Period Counter Register

32-bit current value CTU0 period count register

CT0_PC

Set CTU0 period count register

Period count register (PC) is used in a measurement mode when averaging for multiple periods. It is required because of the high-speed or unstable nature of the incoming signal. Results of the measurement will be accessible only after specified number of periods on the incoming signal are detected. Start of the period is assumed to be a rising edge of the de-bounced input CLKIN line.

0x201C RD – CT0_CRH – CTU0 Capture Register HIGH

This 32-bit register is used to store results of the measurements in $\frac{1}{2}$ or N period measurement modes. In N periods (N is defined by the value stored in the PC register), measurement mode provides accumulated number of 66MHz counts during the positive part of all periods measured.

0x201C WR – CT0_CR0 – CTU0 Set Value of Compare Register 0

32-bit compare register zero (CR0) is used to define shape of the output signal. In all modes except quadrature encoder and measurement modes, counter register CR counts up from the value loaded in LR register and output toggles from low to high when $CR=CR0$. Depending on the other configuration parameters selected, counter may continue count, restart itself, or stop, when value of the CR reaches value stored in CR0 register. CR0 may be used in conjunction with CR1 for the complex PWM waveform generation

0x2020 RD – CT0_CRL – CTU0 Capture Register LOW

32-bit register is used to store results of the measurements in N period measurement modes. In N periods, (N is defined by the value stored in the PC register) measurement mode provides accumulated number of 66MHz counts during the negative (low) part of all periods measured.

0x2020 WR – CT0_CR1 – CTU0 Set Value of the Compare Register 1

32-bit compare register one (CR1) is used to define shape of the output signal. In all modes except quadrature encoder and measurement modes, counter register CR counts up from the value loaded in LR register and output toggles from low to high when $CR=CR0$, then output stay high until $CR0 \leq CR \leq CR1$. Depending on the other configuration parameters selected, counter may continue count, restart itself, or stop, when the value of the CR reaches the value stored in CR1 register. CR1 may be used in conjunction with CR0 for the complex PWM waveform generation

0x2024 WR – CT0_TBR – CTU0 Time-base Divider Register

32-bit TBR (write-only) register defines time-base divider for the time-based capture modes.

5.7.2 FIFO Access**0x1800/0x2028 RD – CT0_FCNTI – CTU0 Input FIFO Count Register**

9-bit, LSB valid, return number of samples available (written from the host to layer) in the input FIFO of the CTU0.

0x1808/0x2030 WR – CT0_FDTI – CTU0 Input FIFO Data Input Register

32-bit write-only register for the input FIFO.

0x1810/0x2034 RD – CT0_FCNTO – CTU0 Output FIFO Count Register

9-bit, LSB valid, returns number of samples available in the output FIFO of the CTU0.

0x1818/0x203C RD – CT0_FDTO – CTU0 Output FIFO Data Input Register

32-bit read-only register for the input FIFO.

0x2040 WR – CT0_IER – CTU0 Interrupt Enable Register

Interrupt generation unit in every CTU is similar to the IGU in PDNA CLI logic except it does not have interrupt mask register for the simpler operation. Interrupt from any of the available sources, if enabled, latched in ISR 0x2040+RD (Interrupt Status register) and forces IRQ request into logic HIGH state. IRQ line remains in HIGH state until all IRQ requests are cleared via ICR 0x2044+WR (Interrupt Clear Register).

The IER register is a bit field in which each bit enables one interrupt source.

The following are the CT0_IER Bit descriptions.

Bit	Name	Description	Reset State
31	DQ_IR_CPT	Request interrupt if counter completes current operation	0
30	DQ_IR_CR0L	Request interrupt if current value of CR < CR0	0
29	DQ_IR_CR0GE	Request interrupt if current value of CR >= CR0	0
28	DQ_IR_CR1	Request interrupt if current value of CR >= CR1	0
27	DQ_IR_LHI	Request interrupt if low-high transition was detected on the input pin (deglitched)	0
26	DQ_IR_LHG	Request interrupt if low-high transition was detected on the gate pin (deglitched)	0
25	DQ_IR_HLI	Request interrupt if high-low transition was detected on the input pin (deglitched)	0
24	DQ_IR_HLG	Request interrupt if high-low transition was detected on the gate pin (deglitched)	0
23	DQ_IR_CRH	Request interrupt if data is available in CRH	0
22	DQ_IR_CRL	Request interrupt if data is available in CRL	0
21	DQ_IR_IFE	Request interrupt if input FIFO is empty	0
20	DQ_IR_IFH	Request interrupt if input FIFO is at least ½ full	0
19	DQ_IR_IFF	Request interrupt if input FIFO is full	0
18	DQ_IR_OFE	Request interrupt if output FIFO is empty	0
17	DQ_IR_OFH	Request interrupt if output FIFO is at least ½ full	0
16	DQ_IR_OFF	Request interrupt if output FIFO is full	0
15-0		Reserved	0

5.7.3 Command Mode **0x2040 RD – CT0_ISR – CTU0 Interrupt Status Register**

This register should be used to define source of the interrupt from the CTU. It will show “1” in the bits that are the source for the interrupt. The ISR keeps its value until cleared by a write to the ICR or by system reset.

CT0_ISR Bit description

The ISR bits match IER.

0x2044 RD – CT0_ICR – CTU0 Interrupt Clear Register

Writing one to any of the bits in ICR will clear matching bit in ISR, thus clearing the interrupt request based on that bit. Note, that if the interrupt condition still exists and is enabled – it will be “fired” again immediately.

CT0_ICR Bit description

The ICR bits match IER/ISR.

Chapter 6 Host / IOM Communication

6.1 Host / IOM Communication Modes

As illustrated in **Figure 6-1**, the PowerDNA API provides four basic ways of communicating between a host and a PowerDNA IOM (cube or RACKtangle):

- DaqBIOS Command API (point-by-point simple I/O, synchronous)
- Buffered I/O in continuous (ACB) or burst (streaming) mode (asynchronous)
- Mapped I/O API (synchronous) — DMap (fixed data size) or VMap (variable data size)
- Messaging — asynchronous, buffered, messaging data format

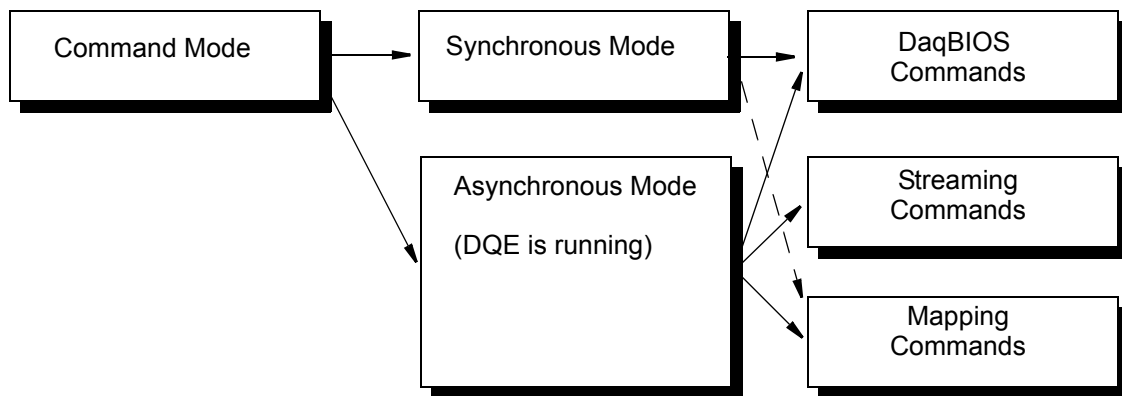


Figure 6-1. Communicating with an IOM

Note that any of the communication modes listed can be selected on a per-I/O board basis and can run independently on the same IOM. Only one API at a time can be used with each I/O board, but each IOM can have multiple I/O boards using the same or different communication modes. If DMap or VMap is selected for more than one board, all such boards are handled as a group.

Some important characteristics of the various modes are:

- In ACB mode, data is transferred in blocks between host and IOM. Each packet contains one block per I/O board configured for ACB operation. If you use multiple ACB I/O boards, you must send separate packets for each such board. Each of the boards can run at a different speed.
- In DMap and VMap modes, data is transferred between host and multiple DMap- or VMap-configured I/O boards on an IOM in a single packet, but you are limited to one data value per channel in each packet. Also, all such boards must run at the same speed, controlled by the IOM clock. Transfer of data between IOM and host is controlled by the host. Update rate of the host maps is usually set at less than half the scan rate of the IOM.

6.2.1 Synchronous vs. Asynchronous

In synchronous modes, also called single scan or simple IO mode, the host sends a request, waits for a reply, and then sends another command. This keeps the host and IOM in lock step. Error detection/correction is handled by the user.

In asynchronous modes, the host sends requests on ticks of a timebase timer, and the software automatically takes care of re-requests when a network collision or loss of a packet occurs. If you prefer, you can work in a manner similar to synchronous mode, sending request after request and processing packets yourself. However, we recommend that you use asynchronous for streaming or data mapping and design your application accordingly.

Asynchronous mode is inherently soft-real-time because collisions on the network cannot be predicted and, therefore, cannot be avoided.

All three APIs (synchronous, buffered, mapped) can be used to communicate with the same IOM, but not at the same time on one I/O board. Once a device on the IOM is switched to an asynchronous mode, you should not issue synchronous commands to that board so as to avoid interfering with any device configuration or timing set up for asynchronous operation.

6.3 Buffered I/O

Buffered I/O modes use temporary intermediate storage to compensate for varying data transfer rates between host and IOM or devices. The two main asynchronous buffered modes are called Advanced Circular Buffer (ACB) and Burst Mode.



6.4 Advanced Circular Buffer (ACB)

As shown in **Figure 6-2**, the Advanced Circular Buffer Mode uses a circular buffer divided into frames. The DaqBIOS engine (DQE) stores data at a known location (the “head”) and reads it at another (the “tail”). When a read or write crosses a frame boundary, the DQE triggers an event.

ACB mode also uses another packet ring buffer for temporary and sequential storage of received packets. When the application detects a missing packet, it requests retransmission of the missing packet and uses the packet ring buffer to place the packet in its proper sequence before writing it to the ACB.

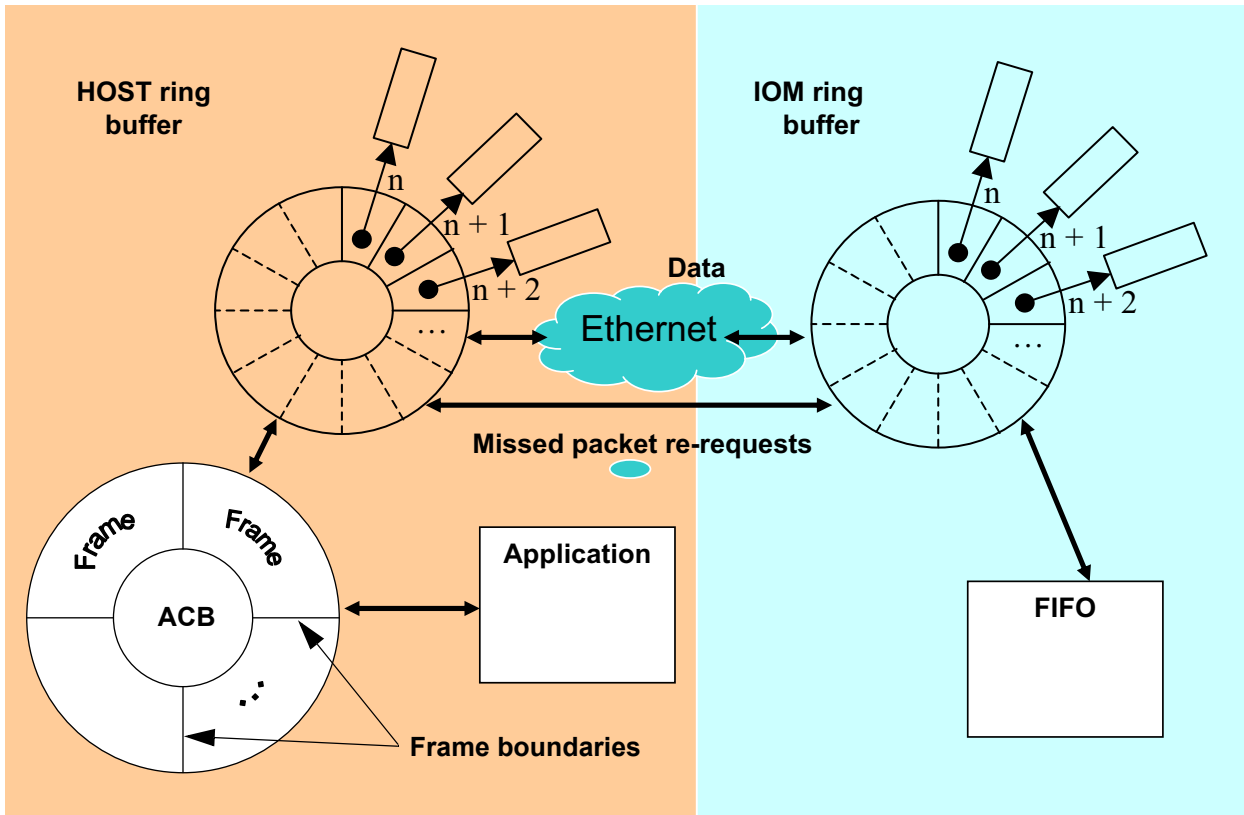


Figure 6-2. Host / IOM Communication in ACB Mode (with DQE)

Once an acquisition is started, DQE stores data into the buffer at a known point (called the head), while the application generally reads data at another position (known as the tail). Both operations occur asynchronously and can run at different rates. However, you can synchronize them either by timer notification or by triggering a DQE event.

To be able to issue a notification to the user application upon receipt of a specific sample or when incoming data reaches a scan-count boundary, DQE segments the buffer into frames. Whenever incoming (or outgoing) data crosses a frame boundary, DQE sends an event to the application. If multi-channel acquisition is performed, the frame size should be a multiple of the scan size to keep pointer arithmetic from becoming unnecessarily complex.

With the ACB, three modes of operation are possible, which differ in the actions taken when the end of the buffer is reached or when the buffer head catches up with the tail.

- In **Single Buffer** mode, acquisition stops when DQE reaches the end of the buffer. The user application can access the buffer and process data during acquisition or wait until the buffer is full. This approach is appropriate when you are not acquiring data in a continuous stream.
- In **Circular Buffer** mode, the head and tail each wrap to the buffer start when they reach the end. If the head catches up to the tail pointer, the buffer is considered full and acquisition stops. This mode is useful in applications that must acquire data with no loss of sample data. Data acquisition continues until either a predefined trigger condition occurs or the application stops DQE. If the application can't keep up with the acquisition process and the buffer overflows, the driver halts acquisition and reports an error condition.
- **Recycled** mode resembles Circular Buffer mode except that when the head catches up with the tail pointer, it doesn't stop but instead overwrites the oldest scans with the new incoming scans. As the buffer fills up, DQE is free to recycle frames, automatically incrementing the buffer tail. This buffer-space recycling occurs whether or not the application reads the data. In this mode, a buffer overflow never occurs. It is best suited for applications that monitor acquired signals at periodic intervals. The task might require that the system digitize signals at a high rate, but not process every sample. Also, an application might need only the latest block of samples.

When the buffer is used for output, the user should fill at least two frames before starting output. Every time a frame becomes empty and ready to accept new data, the DQE triggers an event to the application.

While the ACB may seem a departure from the single and double-buffer schemes you see in most other data acquisition systems, it's actually a superset of them. In Single Buffer mode, the ACB behaves like a single buffer. If configured as a Circular Buffer with two frames, it behaves as a double buffer. With multiple frames, the ACB can function in algorithms designed for buffer queues. The only limitation, which results in more efficient performance, is that the logical buffers in the queues cannot be dynamically allocated or freed and that their order is fixed.

The Ethernet UDP protocol used to transfer data is connectionless and unreliable. Older packets may come first and new packets may never arrive. The ACB assumes that the data comes sequentially without gaps between scans. To accommodate the sequential nature of a data stream with the packet nature of Ethernet, DQE implements an additional intermediate buffer – called the Packet Ring Buffer (PRB), which should not be confused with the separate ACB buffer.

The PRB is a non-contiguous ring buffer intended for data loss recovery. FIFO devices on the IOM send their data to the host in sequentially numbered packets (using the dqCounter field of the DaqBIOS command header). These numbers vary from 0x1 to 0xFFFF and then wrap around (skipping 0). Such numbering allows DQE to notice when a packet is missing — detected whenever a higher-numbered than expected packet is received. (In **Figure 6-2**, if the last packet number was n and we've just received one numbered $n+2$, we know that the packet $n+1$ is missing.) Since the receiving buffer is non-contiguous, we just put the newly arrived packet into the buffer, which was bound to receive it anyway, and send a specific request for the missing one. When it finally arrives, we just put it in its proper place and copy all data into the contiguous ACB in correct order.



A thread transfers data from the ring buffer into the ACB when contiguous chunks of data become available. The data request routine, (`DqGetACBScans()`), also performs additional transfers if a chunk of contiguous data is available at the moment of execution.

6.4.1 Burst Mode

Burst Mode is a streaming mode in which data is sent or received continuously for a specific time duration or until an event such as timer event, buffer full, or buffer empty occurs.

6.5 Message Mode (Msg Protocol)

With messaging devices (serial, CAN, ARINC interfaces), the data is a stream of bytes logically divided into frames, messages, strings, etc. Two characteristics of messaging devices make DMap protocol inefficient, if not impossible, for handling messages and thus generate a need for another protocol specifically designed for messaging. These characteristics are:

1. Since the data is a stream, losing part of the data may change the meaning of the message.
2. Unlike digital or analog data, the timing of data availability depends on the external stream of messages — you cannot predict when and how much data will become available, and whether or not receiving/transmitting errors may exist on the bus.

Messaging layers, therefore, are supported by the **Msg Protocol**, which shares the same buffering mechanism as the ACB protocol. The Msg protocol buffer receives packets and delays releasing newer packets to the user application until it re-requests and receives all the packets in the message stream. Although this protocol does provide a gapless stream of messages, it is not suited for real-time operation because some deadlines may be missed.

Message mode operates in much the same way as ACB mode. The IOM, of course, must have a layer that supports a messaging protocol, such as a CAN-503 layer. When messages are received by the messaging layer, they are stored in the FIFO. As with the streaming version of ACB mode, a messaging layer in Operation mode sends packets (containing the received messages) to the host automatically, without the host having to send a command to request them. When the host receives the message packets, it puts them into a Receiving Message Queue, which is similar to an ACB, and signals an event, which alerts the client program. The client program can then retrieve the messages and process them as needed.

There is also a Sending Message Queue on the host side, into which the client program can insert outgoing messages. These messages are taken from the queue by the reader thread and sent to the IOM. The IOM then transmits the message on the network interface of the layer.

6.5.1 IOM/Host Data Transfer

When the messaging layer receives a message, the message is stored in the FIFO of the layer. When running in Operation mode, the layer checks the FIFO at regular intervals and transmits any as yet unsent messages to the host.

A DQFIFO structure in a packet sent from the IOM to the host may contain one or more messages in its data field. The data field consists of a 16-bit value indicating the size of the next message block, followed by the message block itself, followed by another size value and message block, etc. A size field of 0 terminates the sequence. See the **Figure 6-3** for an illustration.



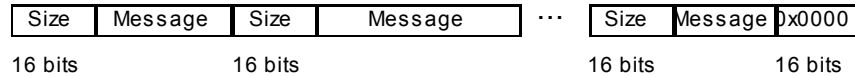


Figure 6-3. Data Field of a RDFIFO Packet Containing Messages

This same format is used to transfer outgoing messages from the host to the IOM for transmission on the network. The host sends a WRFIFO command whose data field holds one or more messages stored the same way.

The format of each message block is specific to the layer type, as described below.

6.5.2 CAN-503 Data Transfer

There are two relevant pieces of information contained in a CAN network protocol packet: the identifier and the message data itself. The identifier is either 11 or 29 bits long, depending on whether it is a standard packet or an extended packet. The data can be 0 to 8 bytes long. In addition, the CAN-503 layer has four network interfaces, which are analogous to channels on other layers. The message data coming from the FIFO of the layer thus has to include the following three pieces of data: the ID, the message data, and the channel that received it. Messages sent to the IOM from the host must also include this information. The message block for a CAN-503 layer is illustrated in **Figure 6-4**.

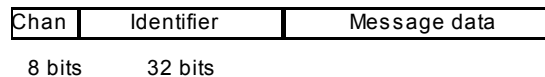


Figure 6-4. Message Block for CAN messages in FIFO

The first byte indicates the channel (network interface), the next four bytes contain the identifier, and the remaining bytes contain the message data. Recall that the size of this block is stored in the 16 bits immediately preceding it, where it appears in an RDFIFO response packet.

6.5.3 PDNALib Structures

The PDNALib requires several data structures to implement Message mode, as described below.

6.5.3.1 Message Struct

The DqMessage struct holds a message. It contains the channel number, the address of the intended recipient, the address of the sender, and the message contents. Like other structures defined in the PDNALib, the message content field is declared last, as a byte array of unspecified size, so that an instance of the struct can be allocated via malloc to any size based on the desired message data size. The address field size is 16 bytes, which is the address size of IPv6. This allows PDNA to support any existing messaging protocol.

6.5.3.2 Message Queue

In PDNALib, the BCB structure can be any of three types: ACB, DMAP, or a third variant called Message Queue (or MSGQ). For each messaging layer installed, the user should create two message queues: one to hold messages to be sent, and one to hold received messages. A Message Queue is similar to an ACB, except that instead of being implemented as a flat byte array, it is a linked list of pointers to DqMessage structures. The sending callback function for Message mode takes DqMessage structures from the sending message queue and converts them to DQ commands, which are then sent to the proper layer on the

IOM. As messages are received by the receiving thread from the IOM, the receiving callback function converts the messages to DqMessage structures, and then stores them in the receiving message queue. In this case, an event is triggered with the `DQ_eDataAvailable` flag set. When the client program gets the event, it can call `DqMsgRecvMessage` to get the message and remove it from the receiving message queue.

Two message queue BCBs must be created for interacting with a message layer: one for sending messages, and one for receiving them. One of two constants must be passed to `DqMsgInitOps` to indicate which direction the BCB is being initialized for.

The DQBCB structure is now able to contain a message queue instead of an ACB or DMAP.

For more detailed information, refer to the PowerDNA Reference Manual API.

6.5.4 Error Recovery

If network problems prevent an occasional message packet from successfully being sent to the IOM or host, PDNLib will attempt to recover by retransmitting or re-requesting the lost packet.

6.5.5 Other Messaging Types

Other types of messages such as SL-501/508, ARINC-429, and MIL-1553 are handled in a manner similar to that of the CAN-503.



6.6 Mapped I/O The basic benefit from using I/O mapping is increased speed and throughput. By maintaining duplicate maps of I/O data in both host and the IOM, both processors can access their own data map(s) as needed, without having to communicate across the network. Communication between host and IOM only has to keep the two maps up to date with each other. UEI offers two types of data mapped I/O: Direct Data Mapping (fixed data size), called DMap, and Variable Data Mapping (variable data size) called VMap, which are described below.

6.6.1 Fixed-Size Data Mapping (DMap) Fixed-size data mapping allocates defined-size (maximum of one packet) areas of input and output data that are continually maintained as mirrors of each other. The following diagram illustrates the structure of DMap operation.

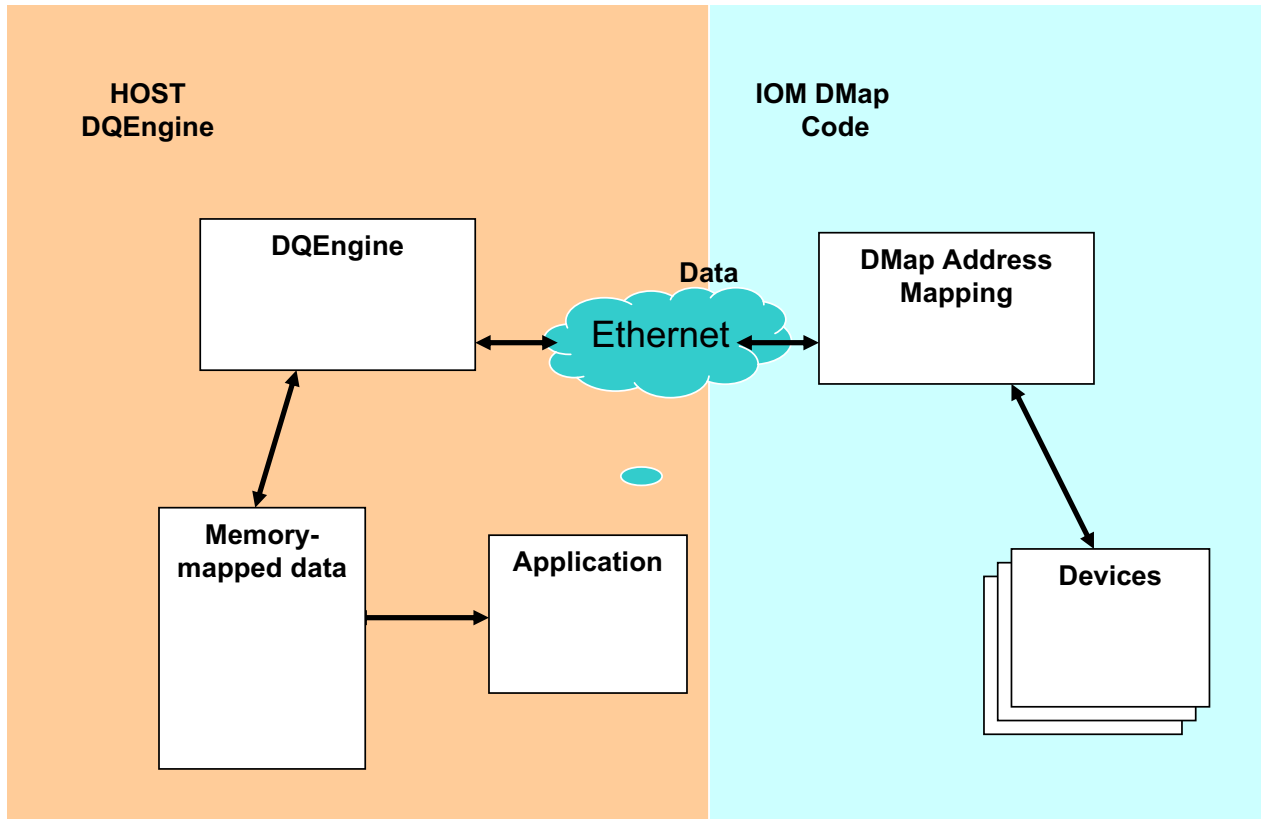


Figure 6-5. Host / IOM Communication in DMap Mode

Each DMap is associated with a device (layer) or group of similar devices in an IOM and each has its own input and output maps. A DMap can store data either in raw or engineering units (volts by default) and can be a maximum of 510 bytes in size (equal to one packet of data). Maps are therefore fully updated as each packet is received from the network.

As indicated in the diagram, input data is acquired by the IOM layers under control of the IOM clock, stored in IOM memory in an area called DMap, and then transferred over the network to the host. In the host, it is stored in the host DMap where it can be accessed by the host application software as needed. Output data is transferred in a similar manner from host to IOM. Packets are transmitted in both directions at a rate determined by the host. The rate is set fast enough to provide a fresh input reading with every reply packet and is typically set at a rate less than half the IOM scan rate (Nyquist rule). The output runs at a rate capable of updating outputs before the next portion of data arrives.

The major attributes of DMap Mode are:

- Each fixed size data map holds a snapshot of simultaneous data for all layers in an IOM that are configured for DMap mode.
- A DMap packet delivers output data from host to IOM: IOM returns most recent input data as a reply.
- Reply is guaranteed within 250 us (133 us with gig-E networks)
- All DMap-configured layers in an IOM are inherently synchronized
- Data is synchronized across multiple IOMs, based on host requests
- All packets are sequentially numbered; the application can re-request lost packets.



6.7.2 Variable-size Data Mapping (VMap)

VMap is another type of mapped I/O that offers variable size data maps. VMap, therefore, is useful for installations in which the size of data to be transferred is unpredictable, such as in messaging or data streaming applications, or when communication bandwidth utilization can be improved by varying the packet size.

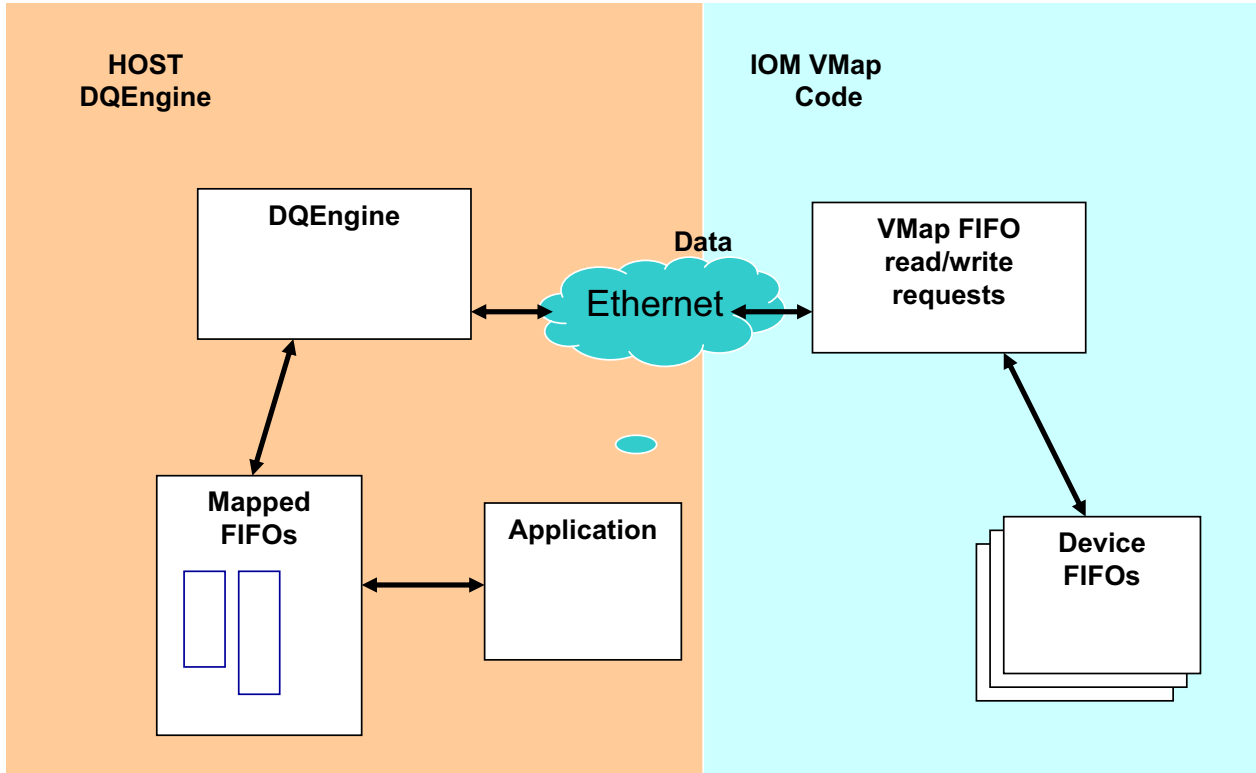


Figure 6-6. Host / IOM Communication in VMap Mode (with DQE)

At high level, VMap is very similar to DMap. A user must create VMap with output and input buffers and add channels/layers of interest to it. As with DMap, DQEngine supports multiple VMaps that can operate at different rates derived from the main DQEngine update period. Unlike DMap, however, VMap packets have additional fields.

First of all, there is a flag field, which is used to guarantee continuity of messaging data. Second, an output buffer adds a pair of fields for each channel in the map at its header. The first field provides the IOM with information on how much data is to be transmitted for that channel and the second field defines the maximum size of data to be received from that channel. Offsets of the output data in the buffer should match the size of the data in the buffer header.



An input packet also contains a flag field as well as the number of bytes actually written, actually received, and (optionally) the number of bytes available in the receive FIFO and the room available in the transmit FIFO. This feature allows flexibility in allocating packet slices for different channels. Each time packets are exchanged between host and IOM, the user application can select different sizes for outgoing and incoming data, taking into consideration the amount of data required to be sent and the size of data accumulated in the receiving FIFO. If you don't use a channel at this time, you should set the "size to send" and the "size to receive" to zero. The header has a fixed width set up before starting VMap operation. The user cannot change the header size on the fly even if the channel is no longer in use.

VMap also has a function that returns the VMap ID to the user for use in multiple IOM installations. Since packets from multiple IOMs may be received by the host out of time sequence, this function gives the host the information necessary to call the right VMap processing routine for that packet.

The packet counter (dqCounter in the DQPKT header) and the flags field work hand-in-hand to synchronize the user application with the DQ Engine.

Table 6-1 lists functions specific to VMap mechanisms. For more detailed information, refer to the PowerDNA API Reference Manual.

Table 6-1 VMap API Functions

Function	Description
DqVmapCreate	This function associates DQE with an IOM and creates the internal structures required to handle VMap operations. The function returns a pointer to BCB to use for all other calls to this VMap and a pointer to memory allocated for the input and output VMap buffers. Each VMap can serve one or more channels of one or more messaging layers (SL-501, CAN-503, 429-566, etc.) located on the same IOM.
DqVmapInitOps	This function must be called, when setting of VMap entries is complete, to finalize it and configure the layer involved. <code>DqVmapInitOps()</code> parses a transfer list, calculates parameters for configuration, channel list, trigger mode, and clocks. Then it completes the transfer lists and prepares the IOM for VMap operation. Notes: A single application can have multiple VMaps created and operated at different update rates. For example, one VMap that controls a device can be updated every 10ms while diagnostic data can be updated every ten seconds. Use the <period> parameter to control the relative update rate as a subdivider of DQ Engine rate.
DqVmapDestroy	This function destroys all memory structures allocated upon DqVmapCreate() and DqVmapAddEntry() calls and stops any ongoing VMap operations associated with this <code>pBcb</code> . Note: It is safe to call this function while VMap operation is running (say, in exception handler).
DqVmapAddEntry	The function adds an entry into the transfer list.

For a description of real time VMap operation without using DQE, refer to "Real-time Variable-size Data Mapping (RtVmap)" on page 106.



6.8 Choosing the Right Layers, Operating System, and Mode

Choosing the right communication mode for your data collection system can significantly improve performance of your system and help meet your design goals. The PowerDNA system offers several choices to meet the needs of your particular application. One of them is sure to meet your particular requirements.

Note that you can select different modes for each layer (but only one mode per layer) and that all DMap- or VMap-configured layers are handled as multi-layer groups.

6.8.1 Attributes of Modes

The various modes and their attributes are described below.

- **Simple I/O Mode (Single Scan)**
 - The major attribute of point-by-point mode is its simplicity and straightforward operation. Requests are sent back and forth between host and IOM in sequence. Error detection/correction is handled by the host.
 - This mode is supported in Windows XP, Linux, Real-time, UEIPAC, and QNX operating systems.
- **ACB Mode**
 - Each subsystem of each layer is handled as a separate stream of data.
 - Every data point is guaranteed delivery, which inherently synchronizes data from different layers.
 - ACB data can be synchronized with other data by using timestamps and/or the SyncX interface between Cubes.
 - In ACB mode, devices can be clocked from external sources.
 - This mode is currently supported in Windows XP and QNX operating systems.
- **Burst Mode (ACB sub-mode)**
 - Stream-to-memory improves performance by storing data into RAM first and then transferring data on a stop trigger.
 - Stream to memory makes 64MB RAM available for temporary storage (equal to 4 seconds of data from four AI-205 layers).
 - Cannot work continuously (limited by size of memory).
 - Can stream data on change of state of digital input.
 - This mode is currently supported in Windows XP and QNX operating systems.
- **Messaging Mode**
 - Uses an ACB to transfer messages.
 - Messages can be grouped together in a single packet to improve performance.
 - Messages can be sent upon receiving a specified amount of data or upon timeout.
 - This mode is currently supported in Windows XP, UEIPAC, and QNX operating systems.



- **DMap Mode**

- Fixed size of data map(s)
- Can be used with both soft and hard real-time systems. With a non-RT OS, the DQE handles lost packet detection/correction and guarantees message continuity. When the real-time set of PowerDNA functions is used on soft or hard real-time OSs, the application software must handle error conditions.
- Single DMap for all DMap-configured layers in Cube or Rack.
- A single DMap packet delivers output data for all DMap layers; IOM returns most recent input data as a reply.
- Reply is guaranteed within 250us (133 us for Gig-E systems).
- Data is synchronized across multiple IOMs based on host requests, guaranteeing that data is synchronized within DMap timebase
- Notifies application if packet is lost and no recovery is available via DQE
- All packets are numbered sequentially; custom application can re-request a lost packet
- This mode is currently supported in Windows XP, Linux, Real-time, UEIPAC. and QNX operating systems.

- **VMap Mode**

- Variable size of data map(s)
- Single VMap for all VMap-configured layers in IOM (IOM can be partitioned into multiple VMaps as needed)
- VMap packet delivers output data and returns input data plus number of samples and number now available.
- VMap packet can be resized dynamically to optimize bandwidth use.
- VMap can be used for AI/DI streaming when soon-to-be-released support is available.
- VMap has built-in mechanism to inform about lost packets.
- All VMap packets are sequentially numbered to maintain message integrity. If packet is lost between host and IOM, IOM will re-output (input) the packet. If lost between IOM and host, IOM will resend the packet.
- This mode is currently supported in Windows XP, Linux, Real-time, UEIPAC. and QNX operating systems.



6.8.2 Application Requirements

Each application has a particular set of requirements, several of which may be opposing. For example, a pure **data acquisition application** may require that all data be delivered without any gaps, but may accept slight delays in delivery of the data. A **control system application**, however, usually requires that all data be delivered on time, but that a few missing data points can be tolerated. In such applications, meeting time deadlines is more important than having gapless data. In **complex applications**, the usual requirement is that all data be delivered in as short a time as possible.

The typical tolerances for delay in data delivery for various types of data collection systems are:

- Data Acquisition System — 1 to 2 seconds
- Control System — 0.5-10 ms
- Complex Application — 10 ms

The tolerances listed above and the bandwidth of the signals being measured affect the choice of communication mode, scan rate, and type of operating system. Some important attributes of desktop and real-time operating systems that influence these choices are:

- **Desktop OS**
 - Windows XP: 10 ms soft real-time
 - Linux 2.6: 10 ms soft real-time (1 ms with pre-emptive patch)
 - All communication modes are supported
- **Real-time OS**
 - Real-time Linux, Windows RTX: 250us hard real-time control loops
 - DMap, VMap, and Single Scan are supported
 - ACB, Msg, and M3 modes are not supported

Mode / OS Support

In making the choice of communication mode for your application, you need to verify that a particular mode is supported by your selected layers, operating system, and also by the OS environment you are working with. **Table 6-2** lists the current state of support offered by UEI for various operating systems. **Table 6-3** shows current UEI support for type of OS environment. **Table 6-4** shows current UEI support for types of Analog Input Layers. **Table 6-5** shows current UEI support for types of DIO and Analog Output Layers. **Table 6-6** shows current UEI support for types of Messaging Layers.

Table 6-2. UEI Support for Various Modes — by Operating System

Operating System	Single Scan	ACB	DMap	VMap	Messaging
Windows XP/Linux	Y	Y	Y	Y	Y
Real-time	Y	N	Y	Y	N
UEIPAC	Y	N	Y	Y	Y
QNX	Y	Y	Y	Y	Y



Table 6-3. UEI Support for Various Modes — by Operating System Environment

OS Environment	Single Scan	ACB	DMap	VMap	Messaging
PDNALib	Y	Y	Y	Y	Y
Framework	Y	Y	Y	N	Y
3rd party drivers	Y	Y	Y	N	Y
Real-time Library	Y	N	Y	Y	N

Table 6-4. UEI Support for Various Modes — by Analog Input Layer

Layer Model No.	Single Scan	ACB	DMap	VMap	Messaging
AI-201	Y	Y	Y	N	--
AI-207/208	Y	Y	Y	N	--
AI-205	Y	Y	Y	N	--
AI-211	Y	Y	Y	N	--
AI-224	Y	Y	Y	N	--
AI-225	Y	Y	Y	N	--

Table 6-5. UEI Support for Various Modes — by Digital IO and Analog Output Layer

Layer Model No.	Single Scan	ACB	DMap	VMap	Messaging
DIO-40X	Y	Y	Y	N	--
DIO-416	Y	N	Y	N	--
DIO-432/433	Y	N	Y	N	--
DIO-448	Y	N	Y	N	--
AO-308	Y	Y	Y	N	--
AO-332	Y	Y	Y	N	--

Table 6-6. UEI Support for Various Modes — by Messaging Layer

Layer Model No.	Single Scan	ACB	DMap	VMap	Messaging
SI-501 / 508	Y	N	N	Y	Y
CAN-503	Y	N	N	Y	Y
429-566	Y	N	N	Y	N
CT-601 / 604	Y	N	Y	N	Y



6.8.3 Selecting the Right Mode for Your Application

To select the communication mode best suited to meet your needs, consider the following selection criteria for each mode:

- **ACB application requirements (typical)**
 - Acquire and store/display data
 - Continuous data > 100 Hz, gapless
 - Data stream faster than 10 kB/s
 - Timing accuracy better than 1/(data rate) seconds
 - Delay between acquisition and delivery is non-critical (0.1s – 1s)
 - IOM controls timing
 - External trigger/clock is required
- **Messaging application requirements (typical)**
 - Send, receive, store a stream of messages
 - Guaranteed message delivery
 - Maximum communication bus loads (serial, CAN, ARINC)
 - Non-critical delay of delivery (within 0.1s-1s)
 - Receive data based on number of bytes, messages, content, timeout
- **DMap application requirements (typical)**
 - Control and simulation applications
 - Host controls timing of data transfers, minimizes response time
 - No network collisions allowed
 - Permits scan rate of 100-500 Hz on non-realtime, 4 kHz on realtime OS
 - Multiple IOM configuration OK
- **VMap application requirements (typical)**
 - Control and simulation applications
 - Variable length messages or Real time data size larger than one scan
 - Host controls timing of data transfers
 - Maximizes IOM performance and bandwidth, minimizes response time
 - Advanced features: message scheduler, frame delays and repetitions

6.8.3.1 Selection Procedure

The general procedure for selecting the communication mode for your system is as follows:

STEP 1: First, define the primary goals of your data collection system.

- Is it a control or data acquisition application?



- What are the signal types, levels, and bandwidths?
- Which is more important – gapless data or timely response?
- Can the application run on a real-time OS?

STEP 2: Choose the I/O layers for your system and select operating parameters for each.

- Signal Type – In/Out, Analog Voltage/Current, Digital Logic Level, Frequency, PWM, Strain, Message (SL, CAN, ARINC, 1553)
- Signal Level
- Bandwidth
- Timing Control (simultaneous or not, Int/ext sync, etc.)

STEP 3: Determine timing requirements and tolerance for gaps in data.

- Data Acquisition — a 1 to 2 second delay is usually acceptable
- Control — 1 0.5 to 10 ms control loop period is typical
- Complex Application — 10 ms control loop data delay is acceptable
- Real-time or Non-Real-Time — can missed deadlines be tolerated?

STEP 4: Select applicable Operating System

- Desktop OS
 - Windows XP: 10 ms soft real-time
 - Linux 2.6: 10 ms soft real-time (1 ms with pre-emptive patch)
 - All communication modes are supported
- Real-time OS
 - Real-time Linux, Windows RTX: 250us hard real-time control loops
 - DMap, VMap, and Single Scan are supported
 - ACB, Msg modes not supported

STEP 5: Verify availability of UEI support for your choices (layers, parameters, data processing, OSs, OS environments, RT/nonRT, messages vs. non-message communication modes). Modify choices as needed.

- See **Table 6-2** to **Table 6-6** starting on page 83

STEP 6: Based on factors listed above, choose Host/IOM communication mode and select optimum parameters.



Chapter 7 How DaqBIOS Protocol Works

7.1 DaqBIOS packet Structure

The DaqBIOS (DQ) protocol relies on the Ethernet protocol for transfer. Current implementation of the IOM firmware allows exchanging DaqBIOS packets over raw Ethernet packets and over UDP packets, but Library implementation under Microsoft Windows™ does not have an option of using raw Ethernet packets.

Ethernet header (14 bytes)	IP header (20 bytes)	UDP header (8 bytes)	DQ header (8 bytes)	DQ data (6-514)	Ethernet CRC (4 bytes)
-------------------------------	-------------------------	-------------------------	------------------------	--------------------	---------------------------

Figure 7-1 DaqBIOS Packet Over UDP Packet

Ethernet header (14 bytes)	DQ header (16 bytes)	DQ data (34-542)	Ethernet CRC (4 bytes)
-------------------------------	-------------------------	---------------------	---------------------------

Figure 7-2 DaqBIOS Packet Over Raw Ethernet Packet

The DaqBIOS protocol relies on the simple concept of acknowledging every packet sent from the host to the IOM.

The DaqBIOS packet header has following fields:

```
typedef struct {
    uint32 dqProlog;          /* const 0xBABAFACA */
    uint16 dqTStamp;         /* 16-bit timestamp */
    uint16 dqCounter;        /* Retry counter +
bitfields */
    uint32 dqCommand;        /* DaqBIOS command */
    uint32 rqId;             /* Request ID - sent from
host, mirrored */
    uint8 dqData[];          /* Data - 0 to 514 bytes
*/
} DQPKT, * pDQPKT;
```

`dqProlog` is always `0xBABAFACA` for revision 2 of the DQ-TS protocol. The DQ-VT protocol available earlier is no longer supported in R2. Instead, we use flow-control and error-correction protocols. The only exception is when you can send a packet with `0xBABAFAC2` as a prolog. In this case, the IOM replies with a proper Prolog and protocol version supported in `dqTStamp`.

`dqTStamp` is a field used for time synchronization between the IOM and the host.

`dqCounter` is used for flow-control between the host and the IOM. The counter starts from one and continues up to 65535, then wraps around.

`dqCommand` is used to specify the command to be executed when sent from the host to the IOM. The host replies with the command executed and with any error flags set. If the IOM processes the command successfully, it replies with the requested command and the `DQREPLY` (0x1000) flag. If the host sends a command with a `DQNOREPLY` (0x2000) flag, the IOM does not send a reply packet.

The following errors located in the upper 16 bits of dqCommand are sent in dqCommand field:

```
/* Masks to extract DQERR_... from command code */
#define DQERR_MASK      0xFFFF0000
#define DQNOERR_MASK   0x0000FFFF

/* The first nybble indicates how the next three
nybbles should be interpreted */
#define DQERR_NYBMASK   0xF0000000 /* general
error/status mask */
#define DQERR_MULTFAIL  0x80000000 /* high bit -
multiple bits indicate error/status */
#define DQERR_SINGFAIL  0x90000000 /* low bit in
first nybble - single error/status */

#define DQERR_BITS      0x0FFF0000 /* error/status
bits or value extracted from here */

/* multiple errors - inclusive or-ed with dqCommand
-- high bit set */
#define DQERR_GENFAIL   0xF0000000 /* general
error/status mask */
#define DQERR_OVRFLW    0x80010000 /* Data
extraction too slow - data overflow */
#define DQERR_STARTED   0x80020000 /* Start trigger
is received */
#define DQERR_STOPPED   0x80020000 /* Stop trigger
is received */

/* single errors/status - not inclusive or-ed bit
0x10000000 set */
#define DQERR_EXEC      0x90010000 /* exception on
command execution */
#define DQERR_NOMORE    0x90020000 /* no more data
is available */
#define DQERR_MOREDATA  0x90030000 /* more data is
available */
#define DQERR_TOOOLD    0x90040000 /* request is
too old (RDFIFO) */
#define DQERR_INVREQ    0x90050000 /* Invalid
request number (RDFIFO) */
#define DQERR_NIMP      0x90060000 /* DQ not
implemented or unknown command */

/*
** The following is reuse of the previous code
** in a different direction: host->IOM
** It means that there was no reply to one
```

```
** of the previous packets of the same type
** Made especially for RDALL, WRRD and RDFIFO
** commands.
*/
#define DQERR_OPS          0x90070000 /* IOM is in
operation state */
#define DQERR_PARAM      0x90080000 /* Device cannot
complete request
/* with specified parameters */

/* network errors */
#define DQERR_RCV        0x90090000 /* packet
receive error */
#define DQERR_SND        0x900A0000 /* packet send
error */
```

`rqId` – request ID. Every time the host sends a packet to IOM, it is accompanied with a new request ID. The Request ID serves to specify what request the reply belongs to when request/reply pairs are overlapped. `RqId` is used under the control of DQE only.

In synchronous operating mode, commands are sent and replies are received. The following picture depicts how the host and the IOM exchange packets under the DaqBIOS protocol:

7.2 DaqBIOS Protocol Versions

To recognize what version of the DaqBIOS protocol the PowerDNA cube supports, the host should send a command with `dqProlog` set to `0xBABAFAC2`. The IOM will reply with the proper prolog and the DaqBIOS protocol version in the `dqTStamp` field and the firmware version in the next four bytes. This sub-protocol allows the host to recognize what version of the firmware is running on the PowerDNA cube and what version of protocol it supports.

7.3 Host and IOM Data Representation

Data on the IOM as well as in the network packets are represented in big-endian format. Data on the PC platform are rendered in little-endian format. Thus, to ensure proper data representation, the user should convert data from network format to host format and back.

7.3.1 Soft and Hard Real-time

We address real-time performance as soft-real-time when timing deadlines are achieved *almost* every time. However, soft-real-time cannot guarantee meeting a deadline in all instances. The majority of general-purpose OSs (Microsoft Windows, Linux, etc.) are soft-real-time with better or worse probability of missing a deadline.

Hard-real-time performance guarantees that no one deadline is missed. Hard-real-time OSs have specially designed schedulers that preempt any ongoing operation when real-time code has to be executed. QNX and RTLinux are examples of hard-real-time OSs.

- 7.3.1.1 Implementation Details** Hard real-time response is achievable only under control of hard-real-time OSs (QNX, for example) or general-purpose OSs with real-time extensions (RTLinux, RTAI Linux.) Real-time OSs are capable of sending DaqBIOS commands to the host without missing deadlines (using DQE). This avoids network collisions completely. Two sets of commands are available for real-time operations: DaqBIOS commands and data mapping commands. Streaming cannot be made real-time because its timing cannot be controlled from the host side.
- If streaming is required under a real-time system, you can implement streaming in FIFO mode rather than streaming mode. FIFO mode assumes that the host sends a request to retrieve data from the IOM side every now and then. This way, the real-time application is responsible for retrieving data on time.
- 7.3.1.2 Immediate and Pending Commands** The firmware processes some commands immediately in the network interrupt vector. Other commands are scheduled and executed by firmware in the pending command thread. A vast majority of DaqBIOS commands are immediate commands. See the PowerDNA API Reference Manual for details. Firmware running on a CM-1 layer sends replies within 200-400 μ s. Commands that include waiting for some hardware events to happen are implemented as pending commands. They include IOCTL calls, setting/getting/saving parameters, and receiving layer capabilities information. The time for pending command execution varies and the user should adjust the timeout prior to calling these commands appropriately.
- 7.3.2 DaqBIOS & Network Security** The PowerDNA Cube may be connected to the Internet, posing virtually no risk to the network it is hosted on. Several features make the PowerDNA Cube next to invulnerable for external attack, in descending order:
1. The PowerDNA Cube has only one UDP open port. By default, this port is 6334 – falling in the IANA unassigned port range 6323-6342. Default security hole scanners will either skip UDP scanning, or skip scans of this range, expecting no useful protocols to run in this range.
 2. The only protocol running on the cube is DaqBIOS – an unpublished protocol with no known exploits. If UDP port 6334 is discovered, it is unusable by anyone who does not understand the protocol.
 3. Commands over the network that involve a change to the IOM memory or settings require a password. Any command that changes internal state of the cube requires user password to be supplied. The password is stored in the encoded NVRAM area of the RTC chip. Any command that changes non-volatile memory requires a super-user password. The password is supplied over DQ protocol.
 4. To prevent disruption of the experiment, the cube has the option to be locked onto an IP/port pair. For compatibility, locking/unlocking is disabled by default. When the locking option is enabled and the host PC establishes communication with the cube, the cube locks on to the host's IP/port pair and will listen for commands only from the locked host – until the host unlocks/releases the cube. Other PCs can only request cube configuration and status requests (e.g., IOM_25431 with AI-201 layer in slot 0 is currently in Locked state).
- Finally, note that the PowerDNA Cube has no known exploitable daemons (e.g., Ms-IIS for http, ftp, etc.)

Chapter 8 DaqBIOS Engine

The DaqBIOS Engine (DQE) is organized as a PowerDNA shared library with which a user application is linked. It is a set of functions and data structures, implementing the DaqBIOS data acquisition protocol. DQE provides all functions necessary to interact with IOMs over the network.

DQE functions are executed within the user process; however, DQE may create additional execution threads for its purposes. Different user applications can use DQE simultaneously. Every process gets its own copy of DQE. DQE implements interlock mechanisms, preventing using of a single IOM by two processes and a single layer in exclusive modes.

DQE is used to simplify PowerDNA programming and shift data contingency and buffering responsibility from a user application to the library.

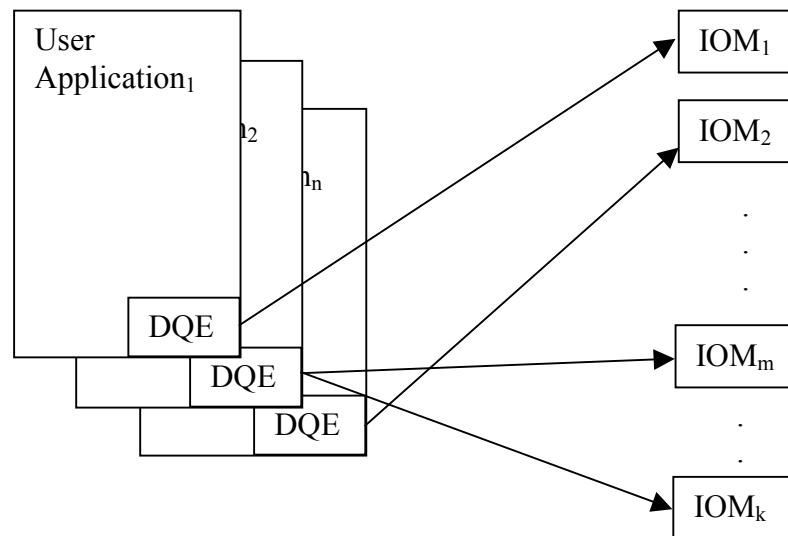


Figure 8-1. User Application/DQE/IOM Interaction.

In the above figure, note that one user application may interact with more than one IOM. The converse is not true.

8.1 Basic Architecture

DaqBIOS Engine consists of the following parts:

- **Sending thread/periodic task (multimedia timer callback under Windows)**

This piece of code periodically wakes up and checks the *command queue* (CQ) of each IOM accessed by the process. It sends one or more commands per IOM per execution cycle and marks it as “waiting for response” so that it isn’t sent the next time. See also *command queue entry* below. There is a single sending thread in every DQE.

- **Receiving Thread**

There is exactly one receiving thread per each IOM. This thread listens to the IOM, receives packets, and routes them to the input buffers according to the IOM’s *command queue*. When a packet arrives from

the IOM, the receiving thread looks up the corresponding entry in the command queue and then relocates the packet to the ring buffer. If there is no corresponding CQ entry, the packet is discarded. If there is a callback associated with the entry, the receiving thread calls it with the specified parameter.

- **IOM Table**

The IOM table is a static array inside the library and is common to all processes. It contains information about all active IOMs being contacted from this host. It includes the list of layers and their options, the processes that are working with them (one process per IOM), and some additional control information. The IOM table access is often made from a critical section.

- **Command Queue**

There is exactly one command queue per IOM. It is a double-linked list that keeps the descriptions (also called command queue entries) of all commands to be sent and all replies to be received to/from the corresponding IOM. The entries are parsed with the *sending thread* and later used by the *receiving thread*. They are put into the queue by `DqSendPkt()` and other DQE calls. The results (after the packets arrive) are used by `Directivity()` calls or DQE callbacks, specified in the command queue entry.

- **Buffer Control Block**

This structure contains control information about Advanced Circular Buffer (ACB) or Data Map (DMap), such as device, subsystem, transfer list, expected byte rate, update period, etc.

- **Reader and Writer Threads**

Reader and writer threads provide transfer of data to and from the packet ring buffer to the ACB or DMap. They are responsible for calling proper data conversion routine depending on the layer type and data format selected. They are also responsible for error correction.

- **Advanced Circular Buffer, Data Map**

These are the data exchangers between the user application and FIFO devices (for ACB) or groups of snapshot devices (for DMap) on IOM.

8.2 Threads and Function

Every instance of DQE has one *sending* and one *receiving* thread. When a process allocates an ACB or a DMap, DQE starts two additional threads. One of them is called *writer* thread and another one *reader* thread. The purpose of these threads is to transfer data from the ACB to the ring buffer for output and from the ring buffer to the ACB for input. The sending or receiving thread wakes the threads up when data needs to be transferred to/from the ring buffer.

8.3 IOM Data Retrieval and Data Conversion

The reader and writer threads call a conversion routine that converts data from the raw format represented in the ring buffer into a floating point representation of volts or other engineering units. If conversion parameters (offset and coefficient) weren't supplied upon creation of ACB or DMap, the data conversion routine converts raw data into native representation – Volts.

Chapter 9 Real-time Operation with an IOM

This section discusses how to perform data mapping and streaming under control of a real-time operating system. The reason for making a separate chapter for real time operation is that writing real-time code can be done more efficiently without using the DQE. Therefore, this section discusses programming of streaming and data mapping operations at low-level.

9.1 Simple I/O

Simple I/O mode, which is commonly associated with lower speed systems, may also be used for real-time applications with a real-time operating system. The key requirement is not speed of operation but rather that all timing be deterministic and that no time deadline be missed.

9.2 Real-time Data Mapping (RtDmap)

Direct data mapping is a mechanism that allows you to create areas of input and output data that mirror data values on the input and output lines of networked IOMs. The following diagram illustrates the structure of DMap operation.

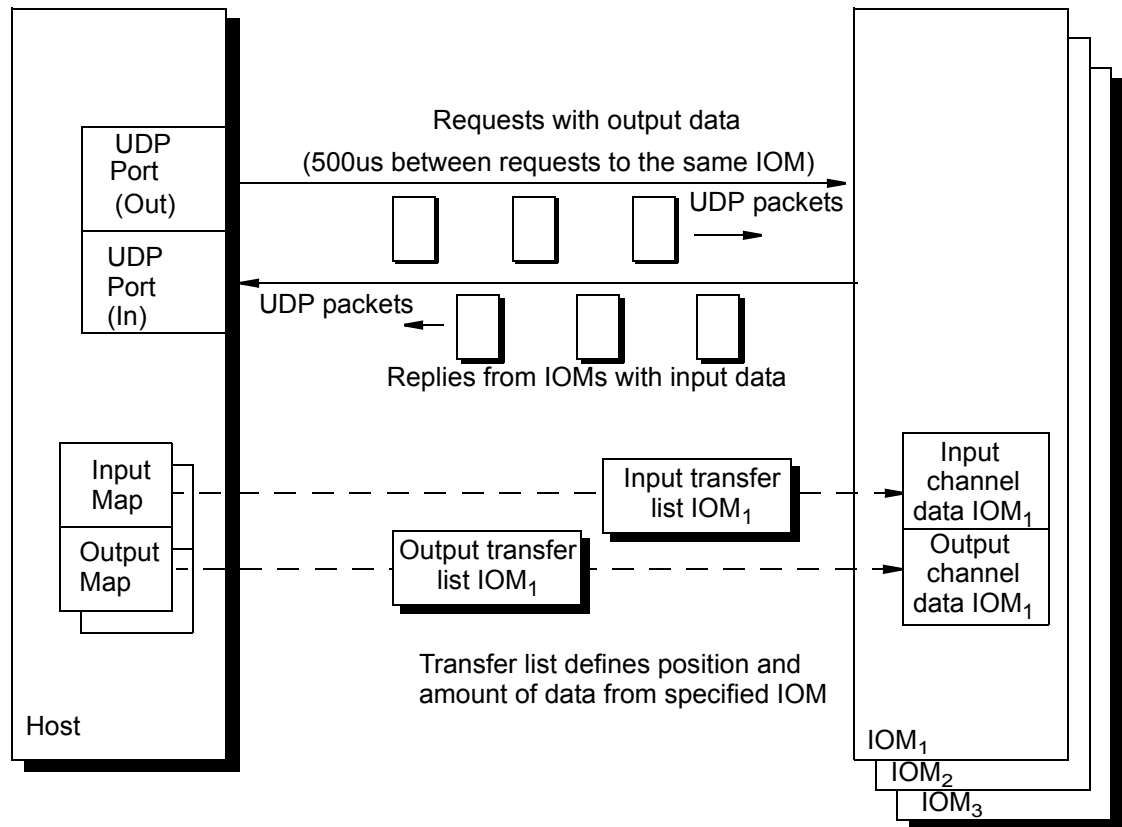


Figure 9-1. DMap Operation

Every DMap has its input and output maps and can work with a single multi-module IOM. Two DMaps can work with the same IOM, but must address different I/O boards (devices) within the IOM.

The maximum size of a DMap is limited to the size of a single packet – 510 bytes, which means that a DMap can be updated by receiving the data contained within a single new packet. Also, DMap allows representation of data either in raw or engineering units (volts by default).

In DMap mode, I/O devices perform at a rate sufficient to update input points fast enough to provide a fresh input reading with every reply packet. The output runs at a rate capable of updating outputs before the next portion of data arrives. Therefore, DMap mode meets the requirements of “hard” real-time operation.

9.2.1 Data Replication over the Network

DMap can be used for input data replication across a local area network if workstation NICs are set into promiscuous mode and receive all reply packets from the UDP interface. DMap can also be used in homogenous networks of IOMs in which IOMs exchange data between each other.

9.2.2 RtDmap Functional Description

The RtDmap API, described in this section, gives easy access to DMap operation without requiring use of the DQEngine. For more detailed information, refer to the PowerDNA Reference Manual.

Operation is as follows:

At each tick of the IOM clock, the IOM firmware scans the configured channels and stores the result in an area of memory called the DMap.

The host PC keeps its own copy of the DMap and synchronizes it periodically with the IOM's version of the DMap. The rate at which the host transfers packets is controlled by the host and is usually set at a rate less than half the scan rate of the IOM clock.

This mode is very useful when the host computer runs a real-time operating system because it ensures that the host refreshes its DMap at deterministic intervals (hard real-time). It optimizes network transfer by packing all channels from multiple I/O boards into a single UDP packet, thus reducing the network overhead.

The standard (non-real-time) low-level API (**DqDmap***** functions) use the DqEngine (DQE) to refresh the DMap at a given rate and to retry a DMap refresh request if, for some reason, a packet is lost. Use of the DQE is necessary on desktop-oriented operating systems to ensure that the DMap is refreshed periodically, but is not required (and not recommended) for use with hard real-time operating systems.

The following is a list of the real time data mapping functions, with short descriptions of each. (Note that each of these functions does not use DQE.)

Table 9-1. RtDMap API Functions

Function	Description
DqRtDmapInit	Initializes the specified IOM to operate in DMAP mode at the specified refresh rate.
DqRtDmapAddChannel	Adds one or more channels to the DMAP.
DqRtDmapGetInputMap	Gets a pointer to the beginning of the input data map allocated for the specified device
DqRtDmapGetInputMapSize	Gets the size in bytes of the input map allocated for the specified device.



Table 9-1. RtDMap API Functions (Forts.)

Function	Description
DqRtDmapGetOutputMap	Gets a pointer to the beginning of the output data map allocated for the specified device.
DqRtDmapGetOutputMapSize	Gets the size in bytes of the output map allocated for the specified device.
DqRtDmapReadScaledData	Reads and scales the data stored in the input map for the specified device. Note: The data read is the data transferred by the last call to <code>DqRtDmapRefresh()</code> . This function should only be used with devices that acquire analog input data such as the AI-2xx series layers.
DqRtDmapReadRawData16	This function reads raw data from the specified device as 16-bit integers. Note: The data read is the data transferred by the last call to <code>DqRtDmapRefresh()</code> . This function should only be used with devices that acquire 16-bit wide digital data such as the AI-4xx series layers.
DqRtDmapReadRawData32	This function reads raw data from the specified device as 32-bit integers. Note: The data read is the data transferred by the last call to <code>DqRtDmapRefresh()</code> . This function should only be used with devices that acquire 32-bit wide digital data such as the DIO-4xx series layers.
DqRtDmapWriteScaledData	This function writes scaled data to the output map of the specified device. Note: The data written is actually transferred to the device on the next call to <code>DqRtDmapRefresh()</code> . This function should only be used with devices that generate analog data such as the AI-3xx series layers.
DqRtDmapWriteRawData16	This function writes 16-bit wide raw data to the specified device. Note: The data written is actually transferred to the device on the next call to <code>DqRtDmapRefresh()</code> . This function should only be used with devices that generate 16-bit wide digital data such as the DIO-4xx series layers.
DqRtDmapWriteRawData32	This function reads raw data from the specified device as 32-bit integers. Note: The data written is actually transferred to the device on the next call to <code>DqRtDmapRefresh()</code> . This function should only be used with devices that acquire 32-bit wide digital data such as the AI-4xx series layers.
DqRtDmapStart	This function starts operation and the IOM updates its internal representation of the map at the rate specified in <code>DqRtDmapCreate</code> .



Table 9-1. RtDMap API Functions (Forts.)

Function	Description
DqRtDmapStop	This function stops operation and the IOM stops updating its internal representation of the data map.
DqRtDmapRefresh	This function refreshes the host's version of the map by downloading the IOM's map. Note: The IOM automatically refreshes its version of the data map at the rate specified in <code>DqRtDMapInit()</code> . This function needs to be called periodically (a real-time OS is necessary) to synchronize the host and IOM data maps.
DqRtDmapRefreshOutputs	This function refreshes the host's version of the map by downloading the IOM's map. Note: The IOM automatically refreshes its version of the data map at the rate specified in <code>DqRtDMapInit()</code> . This function needs to be called periodically (a real-time OS is necessary) to synchronize the host and IOM data maps.
DqRtDmapRefreshInputs	This function refreshes the host's version of the map by downloading the IOM's map. Note: The IOM automatically refreshes its version of the data map at the rate specified in <code>DqRtDMapInit()</code> . This function needs to be called periodically (a real-time OS is necessary) to synchronize the host and IOM data maps.
DqRtDmapClose	This function frees all resources on the specified IOM allocated by the DMAP operation.

**9.2.3 RtDmap
 Typical
 Program
 Structure**

The following is a quick tutorial on use of the RtDmap API (with error handling omitted):

1. Initialize the DMAP to refresh at 1000 Hz.
`DqRtDmapInit(handle, &dmapid, 1000.0);`
2. Add channel 0 from the first input subsystem of device 1.
`chentry = 0;`
`DqRtDmapAddChannel(handle, dmapid, 1, DQ_SS0IN, &chentry, 1);`
3. Add channel 1 from the first output subsystem of device 3.
`chentry = 1;`
`DqRtDmapAddChannel(handle, dmapid, 3, DQ_SS0OUT, &chentry, 1);`
4. Start all devices that have channels configured in the DMAP.
`DqRtDmapStart(handle);`
5. Update the value(s) to be output to device 3.
`outdata[0] = 5.0;`
`DqRtDmapWriteScaledData(handle, dmapid, 3, outdata, 1);`



6. Synchronize the DMAP with all devices.
`DqRtDmapRefresh(handle, dmapid);`
7. Retrieve the data acquired by device 1.
`DqRtDmapReadScaledData(handle, dmapid, 1, indata, 1);`
8. Stop the devices and free all resources.
`DqRtDmapStop(handle, dmapid);`
`DqRtDmapClose(handle, dmapid);`

9.3 Real-time Variable-size Data Mapping (RtVmap)

This feature is similar to RealTime DMap operation (see “Real-time Data Mapping (RtDmap)” on page 93) except that the size of the data transfer is variable.

The RtVmap API, like the RtDmap API, gives easy access to the VMap operating mode without needing the DqEngine.

VMap is a protocol developed for *control* applications in which the ability to get immediate real-time data may be more important than receiving a continuous gapless flow of the data. VMap is also well-suited for many real-time messaging applications, as described below.

Messaging layers are normally supported by the Msg protocol, which shares the same buffering mechanism as the ACB protocol. The Msg protocol buffer receives packets and delays releasing newer packets to the user application until it re-requests and receives all the packets in the previous message stream. Although this protocol does provide a gapless stream of messages, it is not suited for real-time operation because timing is not deterministic.

VMap, however, can provide a real-time alternative to the Msg protocol for messaging devices — at the expense of restricting the ability to recover lost packets. It shifts the decision about whether or not to recover the lost packet to the user application. A set of hard real-time VMap functions is listed below in **Table 9-2**.

At high level, VMap is very similar to DMap. A user creates a VMap with output and input buffers and add channels/layers of interest to it. VMap packets also have additional fields. First of all, there is a flag field required to guarantee continuity of messaging data. Second, an output buffer adds a pair of fields for each channel in the map at its header. The first field provides the IOM with information on how much data is to be transmitted for that channel; the second field defines the maximum size of data to be received from that channel. The offsets of the output data in the buffer should be in agreement with the size of the data in the buffer header.

An input packet also contains a flag field as well as the number of bytes actually written, actually received plus (optionally) the number of bytes available in the receive FIFO, and the room available in the transmit FIFO. This feature allows flexibility in allocating packet slices for various channels. Each time packets are exchanged between host and IOM, the user application can select different sizes for outgoing and incoming data, taking into consideration the amount of data required to be sent and the size of data accumulated in the receiving FIFO. If you don't use a channel at this time, you should set *size to send* and *size to receive* to zero. The header has a fixed width set up before starting VMap operation; the header size cannot be changed on the fly even if the channel is no longer in use.



Note that VMap has a function that returns the VMap ID to the user for use in systems that have multiple IOMs. Since packets from multiple IOMs may be received by the host out of time sequence, this function gives the host the information necessary to call the right VMap processing routine for that packet.

Table 9-2 is a list of the real-time variable data mapping functions, with short descriptions of each. Refer to the PowerDNA Reference Manual API for more detailed information.

Table 9-2. RtVmap API Functions

Function	Description
DqRtVmapInit	Initializes the specified IOM to operate in VMap mode at the specified refresh rate.
DqRtVmapAddChannel	<p>This function adds a channel to <vmapid> VMap. The function adds an entry to the transfer list. Channels with an SSx_IN subsystem are added to the transfer list; channels with an SSx_OUT subsystem are added to the output transfer list.</p> <p>Channel in <cl> should be defined in the standard way including channel number, gain, differential, and timestamp flags.</p> <p>Configuration <flags> for the input subsystem can include DQ_VMAP_FIFO_STATUS to report back the number of samples in the input FIFO waiting to be requested (after output packets are processed). Configuration <flags> for the output system can include DQ_VMAP_FIFO_STATUS to report back the number of samples that can still be written into the output FIFO before it becomes full (after all transmitted bytes have been written). Note that this flag adds a uint16 word to the standard header for an input packet, thus increasing the size of the header and decreasing the size available for data.</p> <p><clSize> specifies the maximum number of array entries.</p> <p>The Output VMap buffer, which transfers data from host to IOM, has the structure shown in Table 9-3 on page 101.</p>



Table 9-2. RtVmap API Functions (Forts.)

Function	Description
<p>DqRtVmapAddChannel (cont.)</p>	<p>The total length of the buffer cannot exceed the size available in the UDP packet minus the combined size of the DQPKT and DQQRD headers.</p> <p>The output buffer of VMap contains information to be written to the channel output FIFOs of the messaging layer (as well as the analog or digital layers equipped with hardware FIFOs). It also specifies the number of bytes to read from the same channel, if any. Data for or from the channel should be assembled in accordance with the message structure of that layer.</p> <p>Flags are used to make data ready and to acknowledge packet execution. This feature arises because VMap relies on continuous data flow compatible with messaging layers as well as continuous acquisition and output and thus must ensure continuity of data. In other words, no message can be sent or received twice.</p> <p>The Input VMap buffer, which transfers data from IOM to host, has the structure shown in Table 9-4 on page 102.</p> <p>The Input VMap buffer contains information showing how much data was actually retrieved from the channel FIFO and how much of the data in the output buffer has been written to that channel.</p> <p>The header size cannot be changed after <code>DqRtVmapStart()</code> is called. In other words, after a channel is added using <code>DqRtVmapAddChannel()</code>, the header size increases by one in the output packet and by one or two (if <code>DQ_VMAP_FIFO_STATUS</code> is set) uint16 words in the input packet. The header allocation cannot be changed until the current VMap is destroyed and a new one is created. If you would like to send zero bytes for that channel or receive zero bytes from a channel, VMap fills the appropriate header field with 0.</p> <p>Note: Each call to <code>DqRtVmapAddChannel()</code> adds one or more transfer list entries. Their indices are zero-origin, sequential, and cumulative. For example, if one adds five channels in the first call to this function, the transfer list index of the last channel is 4. For the next call, the last channel will have transfer list index equal to 9.</p>
<p>DqRtVmapStart</p>	<p>This function sets up all parameters needed for operation – channel list and clock; transfers and finalizes the transfer list. The function also parses the transfer list and stores offsets of the headers for each transfer list entry.</p> <p>If clocked devices (AI/AO) are used, the function programs devices at the rate specified in <code>DqRtDmapInit</code>.</p>
<p>DqRtVmapStop</p>	<p>This function stops operation and the IOM stops updating its internal representation of the data map.</p>



Table 9-2. RtVmap API Functions (Forts.)

Function	Description
DqRtVmapClose	This function destroys the <vmapid> VMap.
DqRtVmapRefresh	<p>This function refreshes the host version of the map by downloading the IOM map.</p> <p>Use the <code>DQ_VMAP_REREQUEST</code> flag if you want to re-request the failed transaction instead of performing a new one. In such case, the <code>dqCounter</code> in the <code>DQPKT</code> header will not be incremented by the host and the IOM will not output/input a new message if the IOM already processed it (reply packet lost). Instead, the IOM will reply with a copy of the previous packet. If the IOM never received the packet, it will process it in the normal way.</p> <p>Note: The IOM automatically refreshes its version of the data map at the rate specified in <code>DqRtVMapInit()</code>. This function should be called periodically (a real time OS is required) to synchronize the host and IOM data maps).</p>
DqRtVmapRefreshOutputs	<p>This function refreshes the host version of the map by downloading the IOM map. Use <code>DQ_VMAP_REREQUEST</code> flag if you want to re-request the failed transaction instead of performing a new one.</p> <p>Note: This function needs to be called periodically (real-time OS is required) to synchronize host and IOM data.</p>
DqRtVmapRefreshInputs	<p>This function refreshes the host version of the map by downloading the IOM map.</p> <p>Note: This function needs to be called periodically (a real-time OS is necessary) to synchronize the host and IOM data maps.</p>
DqRtVmapGetInputPtr	<p>This function gets the pointer to the beginning of the input data allocated for the specified entry.</p> <p>Note: This function can be called only after packet is received.</p>
DqRtVmapGetOutputPtr	<p>This function gets the pointer to the beginning of the output data allocated for the specified entry.</p> <p>Note: This function can be called only after transmission size for all channels is written.</p>
DqRtVmapGetInputMap	<p>Get pointer to the beginning of the input data map allocated for the specified device.</p> <p>Note: This function can be called only after a packet is received, because the actual positions of the input data in the packet for each transfer list entry depend on the number of bytes actually retrieved from the input FIFO. If the number of bytes retrieved is less than requested, VMap will not waste the space in the packet, but rather will pack it to decrease transmission time.</p>



Table 9-2. RtVmap API Functions (Forts.)

Function	Description
DqRtVmapGetOutputMap	This function gets the pointer to the beginning of the output data map allocated for the specified entry. Note: This function can be called only after transmission size for all channels is written. Actual offsets of the data for each channel in the output packet depend on the size of the data stored in the packet header. Thus, this function makes sense only if all data is placed into the packet.
DqRtVmapAddOutputData	This function copies data into the output packet and returns the number of bytes left in the packet. Note: This function modifies the output packet. This function must be called <i>before</i> DqRtVmapRefresh().
DqRtVmapRqInputDataSz	This function requests the number of bytes to receive in the input packet. It returns the number of bytes left in the buffer, the actual size requested, and the pointer to the location where the data will be stored. Note: This function modifies the output packet. This function must be called <i>before</i> DqRtVmapRefresh().
DqRtVmapGetInputData	This function copies data from the input packet and returns the number of bytes copied and the size available in the input FIFO. Note: This function must be called <i>after</i> DqRtVmapRefresh().
DqRtVmapGetOutputDataSz	This function examines the input packet and returns the number of bytes copied from the output packet to the output FIFO and (optionally) how much room is available in the output FIFO. Note: This function must be called <i>after</i> DqRtVmapRefresh().

Table 9-3. Output VMap Buffer

Size	Flags (uint16)
Size to write to Ch0 (uint16)	Size to write to ChN (uint16)
•	•
•	•
•	•
Size to read from Ch0 (uint16)	Size to read from ChN (uint16)
Data for Ch0 (of specified size)	
•	
•	
•	
Data for ChN (of specified size)	



Table 9-4. Input VMap Buffer

Size	Flags (uint16)
No. of bytes retrieved from Ch0 (uint16)	No. of bytes remaining in Ch0 (uint16, optional)
•	•
•	•
•	•
No. of bytes retrieved from ChN (uint16)	No. of bytes remaining in ChN (uint16, optional)
No. of bytes written to Ch0 (uint16)	No. of bytes that can be written to Ch0 (uint16, optional)
	•
	•
	•
No. of bytes written to ChN (uint16 optional)	No. of bytes that can be written to ChN (uint16, optional)
Data from Ch0 (of specified retrieved size)	
	•
	•
	•
Data from ChN (of specified retrieved size)	

**9.3.1 RtVmap
 Typical
 Program
 Structure**

The following is a short tutorial example that uses the RtVmap API (handling of error codes is omitted):

1. Initialize the VMAP to refresh at 1000 Hz:

```
DqRtVmapInit (handle, &vmapid, 1000.0);
```
2. Configure device input output ports using the appropriate DqAdv*** function. For example, the following configures an ARINC-429 device (DEVN) input and output ports 0 to run at 100kbps with no parity and no SDI filtering.

```
DqAdv566SetMode (handle, DEVN, DQ_SS0OUT, 0,
DQ_AR_RATEHIGH | DQ_PARITY_OFF);
DqAdv566SetMode (handle, DEVN, DQ_SS0IN, 0,
DQ_AR_RATEHIGH|DQ_PARITY_OFF|DQ_AR_SDI_DISABLED);
```
3. Add input port 0 to VMAP, set flag to retrieve the status of the input FIFO after each transfer:

```
chentry = 0;
flag = DQ_VMAP_FIFO_STATUS;
DqRtVmapAddChannel (handle, vmapid, DEVN, DQ_SS0IN,
&chentry, &flag, 1);
```

4. Add output port 0 to VMAP, set flag to retrieve the status of the output FIFO after each transfer.

```
chentry = 0;  
flag = DQ_VMAP_FIFO_STATUS;  
DqRtDmapAddChannel(handle, vmapid, DEVN, DQ_SS0OUT,  
&chentry, &flag, 1);
```
5. Enable ARINC-429 ports.

```
DqAdv566Enable(handle, DEVN, TRUE);
```
6. Start all devices that have channels configured in the VMAP.

```
DqRtVmapStart(handle, vmapid);
```
7. Prepare ARINC word to send through port 0 and update VMAP.

```
uint32 arincWord = DqAdv566BuildPacket(data, label,  
ssm, sdi, parity);  
DqRtVmapAddOutputData(handle, vmapid, 0,  
sizeof(uint32), &accepted, (uint8*)&arincWord);
```
8. Specify that we wish to receive up to MAX_WORDS words received by port 0.

```
DqRtVmapRqInputDataSz(handle, vmapid, 0,  
MAX_WORDS*sizeof(uint32), &rx_act_size, NULL);
```
9. Synchronize the VMAP with all devices.

```
DqRtVmapRefresh(handle, vmapid, 0);
```
10. Retrieve the data received by port 0.

```
uint32 recvWords[MAX_WORDS];  
DqRtVmapGetInputData(handle, vmapid, 0,  
MAX_WORDS*sizeof(uint32), &rx_data_size, &rx_avl_size,  
(uint8*)recvWords);
```
11. We can also check how much data was actually transmitted during the last refresh.

```
DqRtVmapGetOutputDataSz(handle, vmapid, 0,  
&tx_data_size, &tx_avl_size);
```
12. Stop the devices and free all resources.

```
DqRtVmapStop(handle, vmapid);  
DqRtVmapClose(handle, vmapid);
```



Appendix A

A.1 Configuring a Second Ethernet Card Under Windows XP

To configure a second Ethernet card for your system, use the following procedure:

A. Set Up Your Ethernet Card (NIC).

If you already have an Ethernet card installed, skip ahead to the next section, "Configure TCP/IP".

If you have just added an Ethernet card, to install it, do the following:

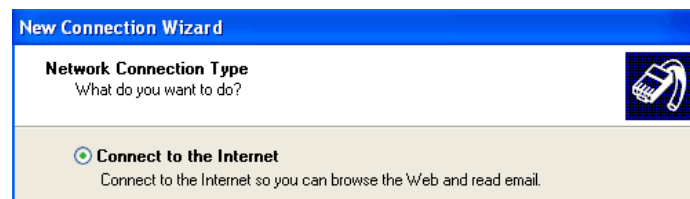
- STEP 1:** From the *Start* menu, select *Control Panel*, and click *Printers and Other Hardware*.
- STEP 2:** From the menu on the left, click *Add Hardware* and follow the on-screen instructions.

NOTE: We recommend that you allow Windows XP to search for and install your Ethernet card automatically. If Windows XP does not find your Ethernet card, you will need to install it manually by following the manufacturer's instructions.

Once your Ethernet card has been installed, continue to the next section.

B. Configure TCP/IP.

- STEP 1:** From the *Start* menu, select *Control Panel*.
- STEP 2:** Under the heading *Pick a Category*, click *Network and Internet Connections*.
- STEP 3:** Under pick a *Control Panel* icon, click *Network Connections*.
- STEP 4:** If you see an icon under *LAN* or *High-Speed Internet* heading for your second NIC, skip ahead to step 10.
- STEP 5:** If there is no icon under *LAN* or *High-Speed Internet* for your second NIC, proceed to step 4.
- STEP 6:** From the menu on the left, click *Create a new connection* to launch the *New Connection Wizard*.
- STEP 7:** Click *Next* and proceed to the *Network Connection Type* window.
- STEP 8:** Select *Connect to the Internet* and click *Next*.

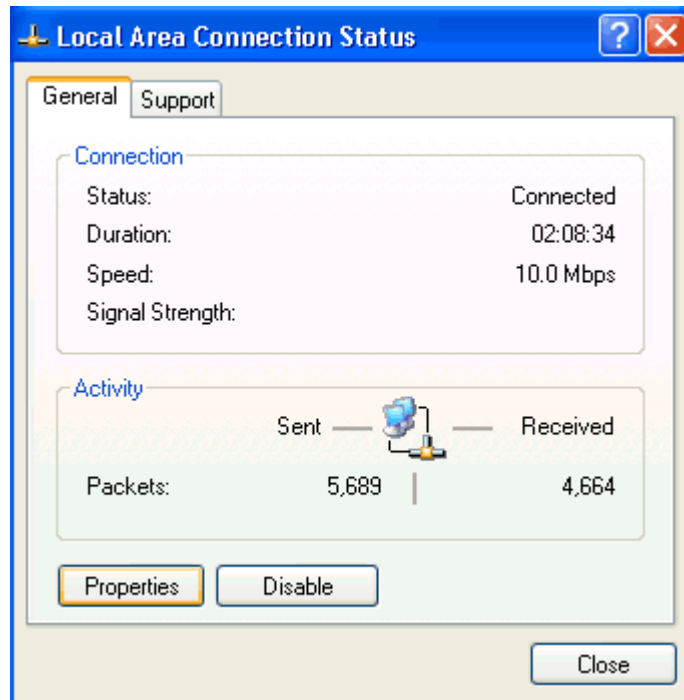


- STEP 9:** Select *Set Up My Connection Manually* and click *Next*.
- STEP 10:** Select *Connect Using a Broadband Connection that is always on* and click *Next*.
- STEP 11:** Click *Finish*.



STEP 12: In the *Network Connections* window, double-click the second icon under *LAN or High-Speed Internet*.

STEP 13: In the next window (see illustration below), click *Properties*.



STEP 14: Click the *General* tab, click once on *Internet Protocol (TCP/IP)*, then click *Properties*.

STEP 15: Click the *General* tab, click *Use the Following IP Addresses*, and in the corresponding boxes, enter 192.168.100.1 for the IP address, 255.255.255.0 for the Subnet Mask, and leave blank the router (or default gateway) information.

STEP 16: Click *Use the Following DNS Server Addresses*.

STEP 17: Make sure the *Preferred DNS Server* box and the *Alternate DNS Server* box are blank.

STEP 18: Click *OK* or *Close* until you return to the *Network Connections* window.

STEP 19: Close the *Network Connections* window.

C. Troubleshooting

If you encounter problems connecting to the network, first check to make sure the Windows XP Internet Connection Firewall is turned off. Follow the instructions below:

STEP 1: From the *Start* menu, select *Control Panel*.

STEP 2: Under the heading *Pick a Category*, click *Network and Internet Connections*.

STEP 3: Under pick a *Control Panel* icon, click *Network Connections*.

STEP 4: Double-click the icon under *LAN or High-Speed Internet*. In the next window, click *Properties*.



- STEP 5:** Click the *Advanced* tab and uncheck the box *Protect My Computer and Network* by limiting or preventing access to this computer from the Internet (see illustration below).



- STEP 6:** Click *OK* or *Close* until you return to the *Network Connections* window.
- STEP 7:** Close the *Network Connections* window.

D. Using the Windows XP Alternate Configuration Setting

If you're using a computer with only one Ethernet port, such as a laptop, you can configure Windows XP to automatically switch settings depending on which network it's connected.

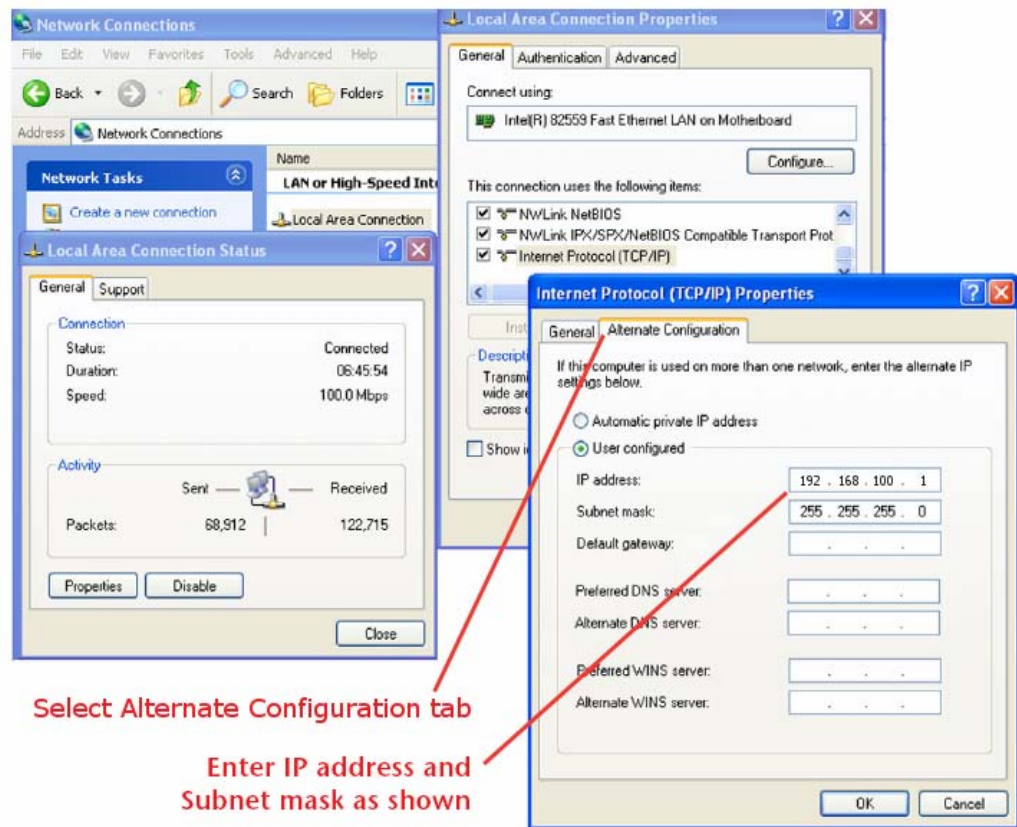
Windows XP users have the ability to configure a second IP address setting under the Control Panel that will allow Windows to pick the correct computer IP setting, based on the device that it finds connected to the Ethernet port. Under this configuration, your primary IP setting is configured for *Obtain IP Address Automatically* for connection to your company Network, and your secondary IP setting (Alternate Configuration) is configured for 192.168.100.1 with a subnet mask of 255.255.255.0 for connection to the PowerDNA cube or DNR-12.

The following steps allow you to configure your alternate IP address, starting at the Control Panel.

- STEP 1:** Double click on *Network Connections*
- STEP 2:** Double click on *Local Area Connections*
- STEP 3:** Click on the *Properties* button
- STEP 4:** Select *Internet Protocol (TCP/IP)* and click on the *Properties* button
- STEP 5:** Select the *Alternate Configuration* tab
- STEP 6:** Select *User Configured*
- STEP 7:** Enter 192.168.100.1 for the *IP address*
- STEP 8:** Enter 255.255.255.0 for the *Subnet mask*
- STEP 9:** Close all open configuration windows using *OK* or *Close*

Use the following screen to configure the *Alternate Configuration* tab located under the Windows XP network configuration screen located in the Windows XP Control Panel.





Once you have this configuration in place, your computer will look for the attached device on your Ethernet port during “Boot Up” or during a Windows “Log On” operation. If it sees a powered on PowerDNA cube connected to the Ethernet port, it will automatically switch to using the secondary IP address. If the computer sees a DHCP network connected to the Ethernet port, it will use the primary IP configuration and negotiate an IP address with your company network as required.

If you are in the office and you want to check your server email: Plug in the Ethernet cable for your company’s network connection into your computer and either power up your computer and log onto the network as you normally do, or if your computer is already powered on, perform a Windows “Log Off” and then a “Log On” and log onto your company network as you normally do.

If you are working in the field with a PowerDNA cube or DNR-12: Plug in the Ethernet cable from the data acquisition system into your computer and make sure that the data acquisition system is powered on. Then, either power up your computer and bypass your network log on screens, or if your computer is already powered on, perform a “Log Off” and then a “Log On” and bypass your network logon screens.

A.2 Configuring a Second Ethernet Card Under Windows 2000

This section describes procedures for configuring a second Ethernet Card under Windows 2000.

The procedure is as follows:



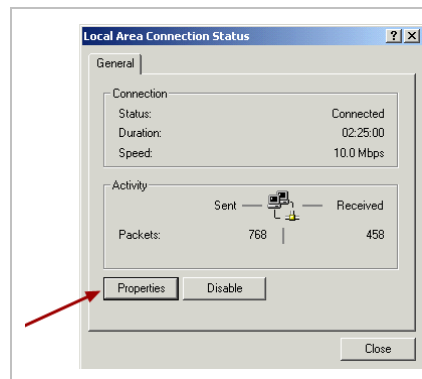
A. Set Up Your Ethernet Card (NIC)

Windows 2000 will normally detect and install your Ethernet card and TCP/IP automatically. To check that your card has been installed, run through the following steps.

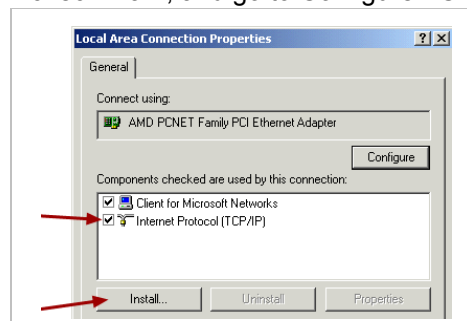
- STEP 1:** From the *Start* menu, select *Settings* and then select *Network and Dial-up Connections*.
- STEP 2:** If you see a *Local Area Connection* icon, your Ethernet card has been detected and installed, skip ahead to the section *Configure TCP/IP*. If you do not see this icon, proceed to step 3.
- STEP 3:** From the *Start* button, select *Settings*, then *Control Panel*. Double-click on the *Add/Remove Hardware* icon and follow the on-screen instructions. We recommend that you allow Windows 2000 to search for and install your Ethernet card automatically. If Windows 2000 does not find your Ethernet card, you will need to install it manually by following the manufacturer's instructions.
- STEP 4:** Once your Ethernet card has been installed, click *OK* and continue with the next section.

B. Install TCP/IP

- STEP 1:** From the *Start* menu, select *Settings* and then select *Network and Dial-up Connections*.
- STEP 2:** In the *Network and Dial-up Connections* window, double-click on the *Local Area Connection 2* icon
- STEP 3:** In the *Local Area Connection 2 Status* window, click *Properties*:



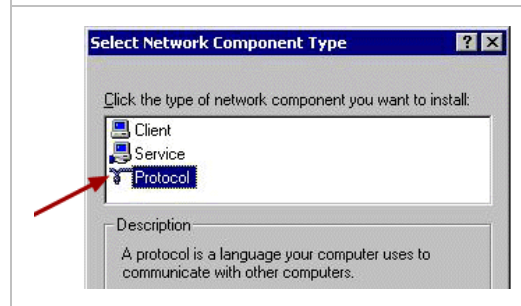
- STEP 4:** If *Internet Protocol (TCP/IP)* is listed, make sure the box next to it contains a check mark, and go to *Configure TCP/IP*.



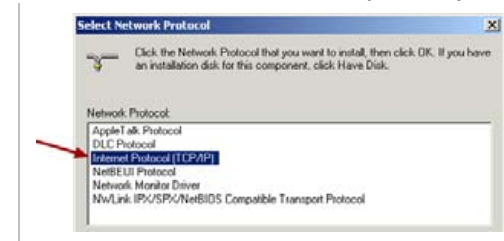
- STEP 5:** If *Internet Protocol (TCP/IP)* is not listed, click on *Install*.



STEP 6: In the next window, double click on Protocol..



STEP 7: Select Internet Protocol (TCP/IP), and click OK.



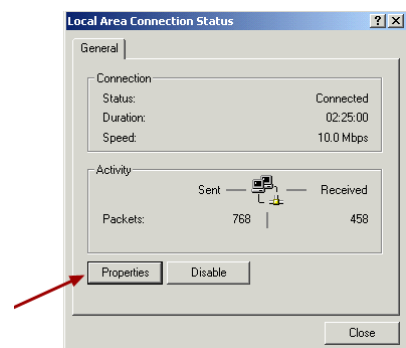
STEP 8: Make sure the box beside *Internet Protocol (TCP/IP)* contains a check mark, and proceed to the next section, *Configure TCP/IP*.

C. Configure TCP/IP

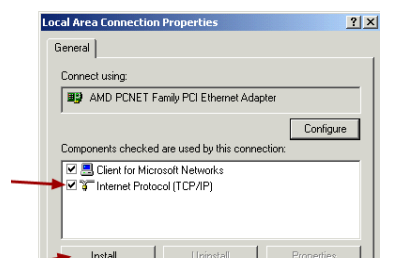
STEP 1: From the Start menu, select Settings and then select Network and Dial-up Connections.

STEP 2: In the Network and Dial-up Connections window, double-click on the Local Area Connection 2 icon.

STEP 3: In the Local Area Connection 2 Status window, click Properties:



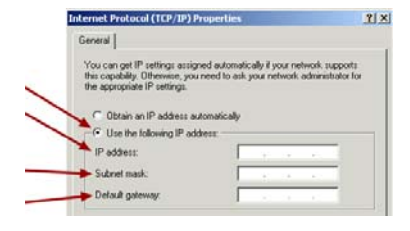
STEP 4: Click once on Internet Protocol (TCP/IP). Then click Properties.



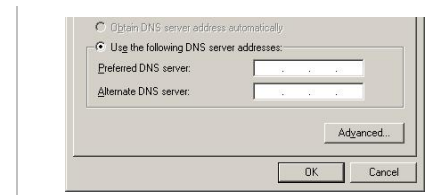
STEP 5: Select *Use the following IP address*, and type 192.168.100.1

In the Subnet mask box, type 255.255.255.0.

Leave the Default Gateway box blank.



STEP 6: Select *Use the following DNS server addresses* and: Make sure the *Preferred DNS server* box and the *Alternate DNS server* boxes are blank.



STEP 7: Click *OK*, click *OK* in the *TCP/IP Properties* window, click *OK* in the *Local Area Connection* window and click *Close* in the *Local Area Status* window.

STEP 8: Close the *Network and Dial-up Connections* window.

A.3 Configuring a Second Ethernet Card Under Windows NT

A. Set Up Your Ethernet Card (NIC)

If you installed your Ethernet interface before (or at the same time as) you installed Windows NT, then the system should have automatically detected it and you should proceed to the next section, "Install and Configure TCP/IP." Optionally, you may follow steps 1-3 below to confirm that your interface is recognized.

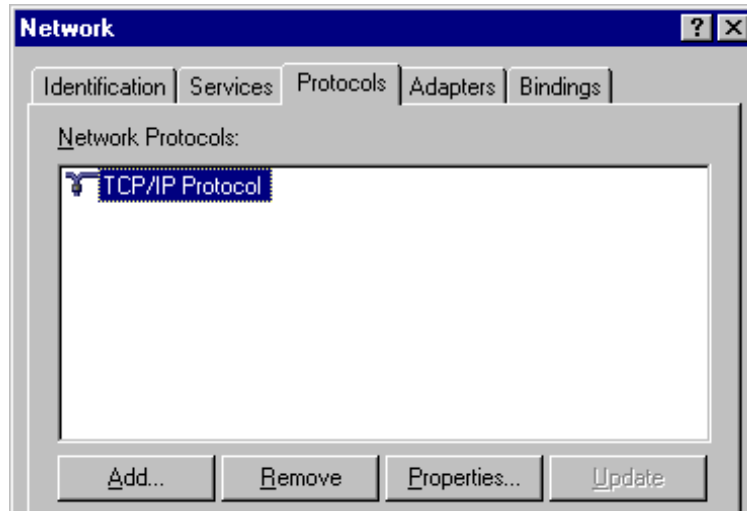
If you obtained an Ethernet interface after Windows NT was already on your computer, do the following:

- STEP 1:** From the Start menu, select *Settings* and then select *Control Panel*.
- STEP 2:** Double-click on the *Network* icon.
- STEP 3:** Click on the tab labeled *Adapters*. You should then see an entry for your Ethernet card. If you do not see one, continue to step 4 to install it. Otherwise, click *OK* and skip ahead to *Install and Configure TCP/IP*.
- STEP 4:** Click *Add...* and follow the on-screen instructions. Select your Ethernet card from the list shown, or, if it is not included in the list, click *Have Disk...* and insert the diskette that came with the card. Even if your card does appear in the list, it's a good idea to use the diskette to make sure you have the latest drivers.
- STEP 5:** Restart your computer if Windows gives you the option to do so. Wait for the system to restart before continuing with the next section.



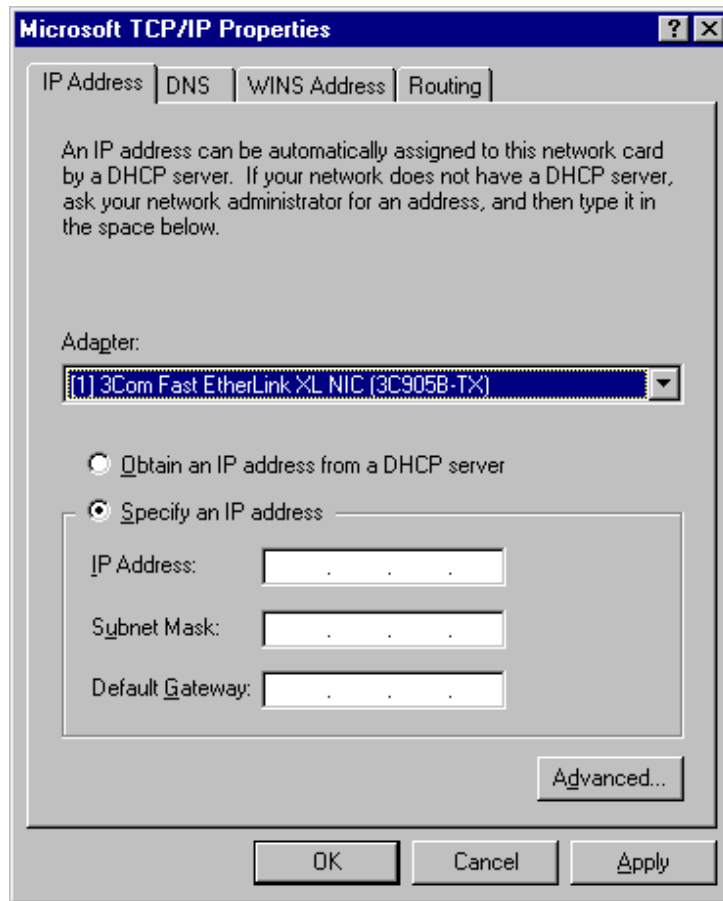
B. Install and Configure TCP/IP

- STEP 1:** From the Start menu, select Settings and then Control Panel.
- STEP 2:** Double-click on the Network icon, then click the Protocols tab.
- STEP 3:** In the list of Network Protocols, look for TCP/IP Protocol. If you don't see it, click Add..., select TCP/IP Protocol, and then click OK.
- STEP 4:** Select TCP/IP Protocol in the list of Network Protocols and then click Properties... A Microsoft TCP/IP Properties window will open.



- STEP 5:** Click on the IP Address tab if it is not already selected.
- STEP 6:** Make sure that the radio button next to Specify an IP address is selected.
- STEP 7:** Enter 192.168.100.1 for IP Address, 255.255.255.0 for Subnet Mask, and leave blank the Gateway Address (in the Default Gateway box.)





STEP 8: Click on the DNS tab.

Leave blank the Host Name and Domain fields.

STEP 9: Click OK to close the Microsoft TCP/IP Properties window.

STEP 10: Click Close to close the Network control panel.

STEP 11: Restart your computer.

STEP 12: You should now be able to access network-based services.

A.4 Configuring a Second Ethernet Card Under Windows 95/98/SE/ME

A. Set Up Your Ethernet Card (NIC)

If you installed your Ethernet card before (or at the same time as) you installed Windows 95/98/ME, then the system should have automatically detected it and you should proceed to the next section, Install TCP/IP. Optionally, you may follow steps 1-3 below to confirm that your card is recognized.

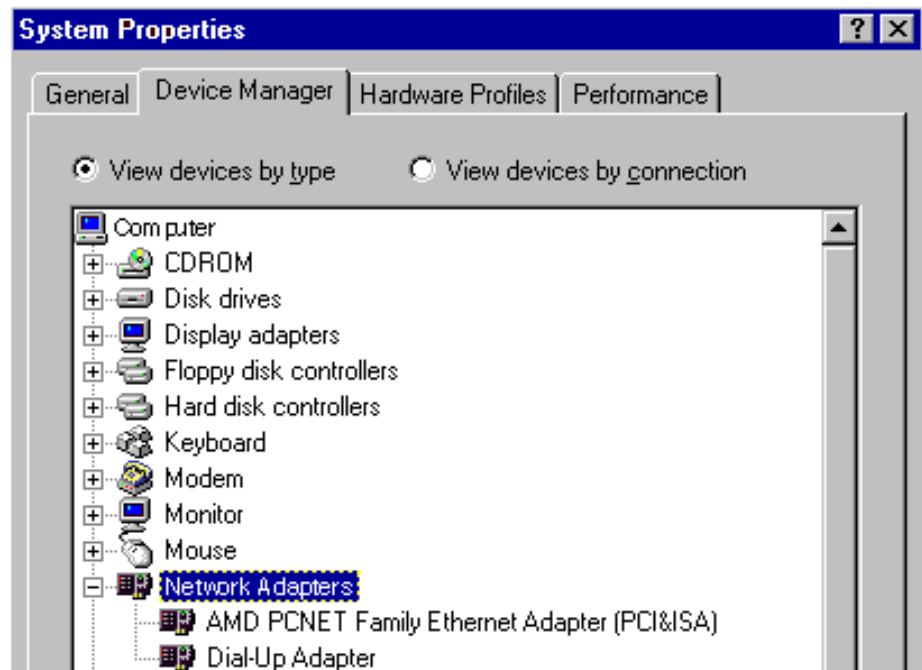
If you obtained an Ethernet interface after Windows 95/98/Me was already on your computer, then do the following:

STEP 1: From the Start menu, select Settings and then select Control Panel.

STEP 2: Double-click on the System icon, then click on the tab labeled Device Manager.



- STEP 3:** Double-click on Network adapters to display a list of the network interfaces that are installed on your computer. If you see two entries other than the Dial-Up Adapter, one is your second Ethernet card. Skip ahead to Install TCP/IP. If you do not see your second Ethernet card, continue to step 4 to install it.



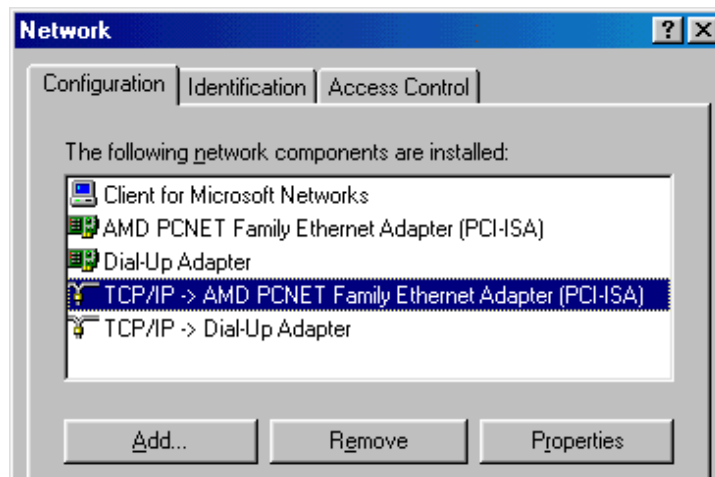
- STEP 4:** If an entry for your second Ethernet card appears here, you probably do not need to run any software included with your card, but keep the software handy just in case you need it later to resolve a problem.
- STEP 5:** Note the name of your second Ethernet card.
- STEP 6:** Close the System Properties window (the Control Panel window should still be open).
- STEP 7:** Open the Add New Hardware control panel and follow the on-screen instructions. We recommend that you allow Windows to search for and install your card automatically.
- STEP 8:** Restart your computer if Windows gives you the option to do so. Then continue with Install TCP/IP.

B. Install TCP/IP

To determine whether TCP/IP software is already installed on your computer, follow these steps:

- STEP 1:** From the Start menu, select Settings and then Control Panel.
- STEP 2:** Double-click on the Network icon. Click on the Configuration tab if it is not already selected.





STEP 3: Look in the box labeled The following network components are installed.

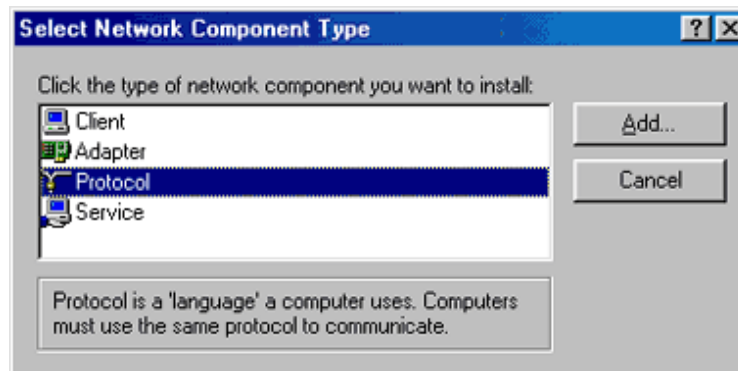
STEP 4: If you see IPX/SPX-compatible Protocol or NetBEUI in the list, select it, then click the Remove button to delete it. These protocols are used by some networked applications, especially games, but they may interfere with your Ethernet connection.

STEP 5: If you don't see TCP/IP for your second Ethernet card, then continue with step 4. If you do see TCP/IP for your second Ethernet card, skip ahead to Configure TCP/IP.

Do these steps only if you do not see TCP/IP listed in your Network control panel for your second Ethernet card.

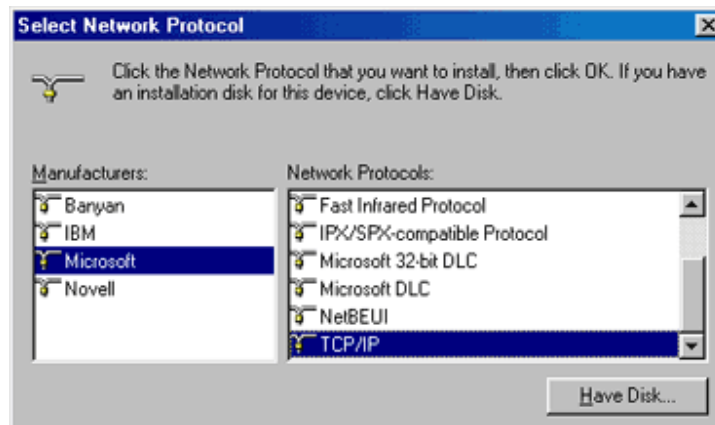
STEP 6: In the Network control panel, click the Add... button.

STEP 7: In the Select Network Component Type window, choose Protocol and click the Add... button.



STEP 8: In the Select Network Protocol window, select Microsoft under Manufacturer and TCP/IP under Network Protocols.

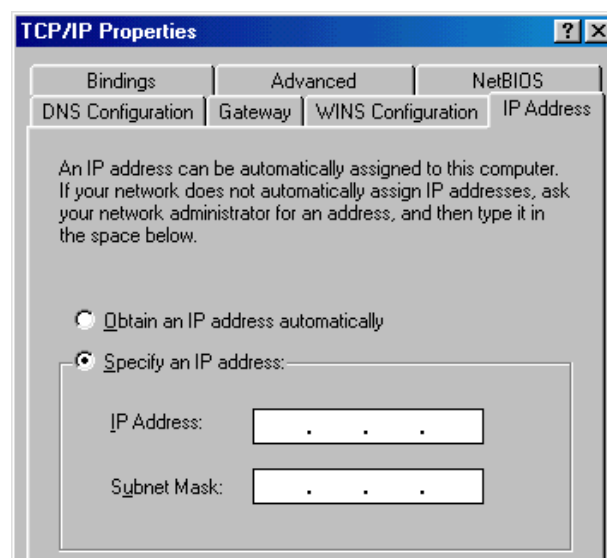




- STEP 9:** Click the OK button to return to the Network control panel, then click the OK button again to exit the control panel.
- STEP 10:** Restart your computer if Windows gives you the option to do so. Then continue with Configure TCP/IP.

C. Configure TCP/IP

- STEP 1:** From the Start menu, select Settings and then Control Panel. Double-click on the Network icon. Click the Configuration tab if it is not already selected.
- STEP 2:** In the box labeled The following network components are installed, select TCP/IP. TCP/IP is listed at least twice, so choose the one followed by the name of your second Ethernet card (do not choose TCP/IP -> Dial-up Adapter).
- STEP 3:** Click the Properties button.
- STEP 4:** In the TCP/IP Properties window, click on the IP Address tab.
- STEP 5:** Make sure that Specify an IP address is selected.
- STEP 6:** Enter 192.168.100.1 for IP Address and 255.255.255.0 for Subnet Mask.

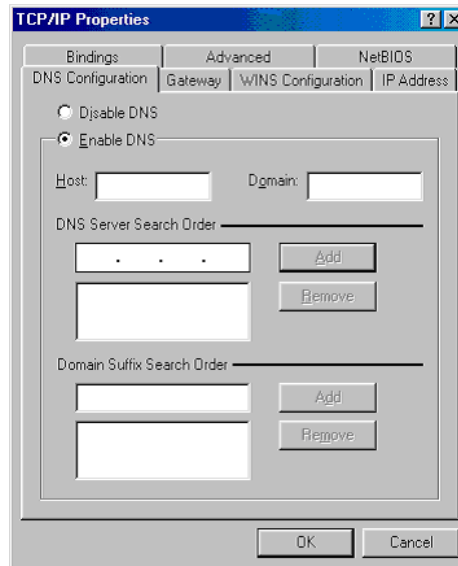


- STEP 7:** Click on the DNS Configuration tab.



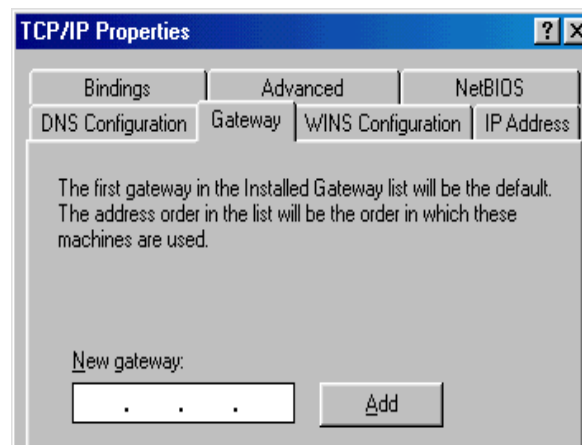
STEP 8: Select Enable DNS.

Make sure the Host and Domain information is blank.



STEP 9: Click on the Gateway tab.

Make sure the box labeled New gateway is blank.



STEP 10: Click the OK button to return to the Network control panel.

STEP 11: Click OK to exit the Network control panel.

STEP 12: Restart your computer if Windows gives you the option to do so.



Index

C

- C class network 11
- Configuring a Card Under Win 95/98/SE/ME 112
- Configuring a Card Under Windows NT 110
- Configuring a Card Under Win XP 104

D

- DHCP 11
- DIN rail 20
- DNA-DR 20
- DNA-PSU-24 7
- Documentation 3

F

- Field connections 20
- Firmware
 - updating 17
- Front-panel layout 9

G

- Gateway
 - mask 11

H

- Host / IOM Communication Modes 70

I

- I/O layers
 - modifying 23
- IP address
 - default 10
 - modify 11
 - modifying 10

M

- Messaging 70
- Mounting 20

N

- Network mask 11
- Network performance
 - improving 11

P

- Power supply
 - for Cubes 7
- PowerDNA Explorer 24

R

- Real-time Operation 93
- Repairs 23
- Reset button 19

S

- Self-diagnostics 9
- Setup program 7
- show command 10
- Subnet 13
- Synchronous and Asynchronous Modes 71

T

- Terminal-emulation program 9

U

- Upgrades 23

W

- Windows
 - Registry 8