



The High-Performance Alternative

DNR-12-1G

RACKtangleTM Data Acquisition System

User Manual

March 2008 Edition
PN Man-DNR-12-1G-0308
Version 1.0

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 infringement 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 the UEI 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 Electronics 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 The DNR-12-1G RACKtangle System	1
1.1 Organization of Manual	1
1.2 PowerDNR DNR-12-1G System	3
1.3 Specifications	4
1.4 Key Features	5
1.5 DNR-12-1G PowerDNR System Enclosure	6
1.5.1 Cooling Air Flow	7
1.5.2 DNR Power, CPU/NIC, and I/O Modules	8
1.6 DNR-POWER-DC Module	12
1.7 DNR-CPU/NIC Module	17
1.8 DNR-Buffer Module	18
1.9 DNR-IO-Modules	18
Chapter 2 Installation and Configuration	19
2.1 Initial Installation Guide	19
2.1.1 Inspect Package	19
2.1.2 Install Software	19
2.2 Initial Boot-up	21
2.2.1 IP Addresses on the IOM	23
2.3 Improving Network Performance	25
2.3.1 Troubleshooting	29
2.4 PowerDNA Explorer Quick-Start	30
2.4.1 Updating Firmware	32
2.4.2 Firmware Update Instructions	33
2.5 Mounting and Field Connections	35
2.5.1 Physical Dimensions	35
2.6 Wiring	36
2.7 Peripheral Terminal Panel Wiring	37
2.8 Repairing (and Upgrading) Your DNR-12	37
2.9 Configuring a NIC Port for Diagnostic Mode	37
Chapter 3 PowerDNA Explorer	43
3.1 The Main Window	43
3.2 Menu Bar	43
3.2.1 File Menu	43
3.2.2 Network Menu	44
3.2.3 View Menu	46
3.2.4 Help Menu	46
3.2.5 Toolbar	46
3.2.6 Device Tree	46
3.2.7 Settings Panel	47
3.2.8 Digital Input/Output Module Settings	49



3.3	Analog Output Module Settings	54
3.4	Analog Input Module Settings	55
3.5	Counter/Timer Module Settings	56
Chapter 4 The DNR-CPU-1000 Core Module		59
4.1	Device Architecture of DNR Core Module	60
4.1.1	Primary Network Interface MII Port – NIC1	61
4.1.2	Diagnostic Network Interface Port – NIC2	61
4.1.3	RS-232 Port	61
4.1.4	USB 2.0 Dual Port (Controller and Slave)	61
4.1.5	32MB Flash Memory	61
4.1.6	128MB of SDRAM (Upgradable to 1 Gb	61
4.1.7	SYNC Port	61
4.1.8	SD Card	61
4.1.9	LEDs	61
4.1.10	Watchdog Timer With Real-time Clock (Battery Backed)	61
4.2.11	CPU / NIC Pinouts	62
Chapter 5 Programming Module-specific Functions		63
5.1	Overview	63
5.2	Startup Sequence	63
5.3	Setting Core Module Parameters	64
5.3.1	Setting Parameters Via Serial Interface	67
5.3.2	Clock and Watchdog Access	69
5.4	Common Module (Layer) Interface	70
5.4.1	Channel List	70
5.4.2	Configuration Flags	72
5.4.3	EEPROM User Area Access	73
5.4.4	PowerDNA Module Signaling	75
Chapter 6 Host / IOM Communication		78
6.1	Host / IOM Communication Modes	78
6.1.1	Additional Modes	78
6.1.2	Synchronous and Asynchronous Modes	79
6.1.3	Buffered I/O	79
6.1.4	Burst Mode	79
6.2	Advanced Circular Buffer (ACB)	79
6.3	DMap Structure and Function	83
6.3.5	DMap Functional Description	84
6.4	Variable-size Data Mapping Functions (VMap)	85
6.5	Message Mode	87
6.6	IOM Operating Modes	88
6.6.1	Initialization Mode	89
6.6.2	Configuration Mode	89
6.6.3	Operation Mode	90
6.6.4	Shutdown Mode	90
6.7	Heartbeat Feature	91
6.7.1	Heartbeat and Safe States	91



6.7.2	Heartbeat and Moving Token Mechanism	91
6.7.3	Heartbeat Processing in Sending Thread	91
Chapter 7	DaqBIOS Protocol	93
7.1	DaqBIOS Packet Structure	93
7.2	DaqBIOS Protocol Versions	95
7.3	Host and IOM Data Representation	95
7.3.1	Soft and Hard Real-time	95
7.3.2	DaqBIOS & Network Security	96
Chapter 8	DaqBIOS Engine	97
8.1	Basic Architecture	97
8.2	Threads and Functions	98
8.3	IOM Data Retrieval and Conversion	99
Chapter 9	Real-time Operation with an IOM	100
9.1	Real-time Data Mapping (RtDmap) Functions	100
9.2	Real Time Variable-size Data Mapping (RtVmap)	103
9.2.1	Real-time Operation: in Receiving Thread	108
9.3	Synchronous Mode Programming	109
Appendix A.		110
A.1	Configuring a Second Ethernet Card Under Windows XP	110
A.2	Troubleshooting	111
A.3	Using the Windows XP Alternate Configuration Setting	112
A.4	Configuring a Second Ethernet Card Under Windows 2000	113
A.5	Configuring a Second Ethernet Card Under Windows NT	116
A.6	Configuring a Second Ethernet Card Under Windows 95/98/SE/ME	118
Appendix B	Replacing Fuses	123
Index		125



Table of Figures

Chapter 1 DNR-12 RACKtangle™ System	1
1-1 UEI Typical PowerDNR DNR-12 RACKtangle% System	3
1-2 Technical Specifications	4
1-3 PowerDNR Product Features	5
1-4 Typical PowerDNR DNR-12-ENCL Enclosure (Exploded View)	6
1-5 DNR-12 Air Flow	7
1-6 PowerDNR System Front Panel Arrangement	8
1-7 DC Power Module LEDs	9
1-8 DNR-CPU-1000 Module LEDs	9
1-9 Typical I/O Module LEDs	10
1-10 DNR-POWER-DC Module	13
1-11 DNR Filler Panel for empty slots	14
1-12 DNR Ejection/Insertion Lever Operation	14
1-13 DNR-BRACKET Reversible Mounting Bracket	15
1-14 Functional Block Diagram of DNR-POWER Module	16
1-15 Functional Block Diagram of DNR-12-ENCL	17
Chapter 2 Installation and Configuration	19
2-1 Typical MTTTY Screen	22
2-2 Show System Configuration	23
2-3 Single DNR-12-1G Direct-Connected to Host without LAN Switch	25
2-4 Single Network for Both Operation and Diagnostics with 2 DNR Racks and Switch	26
2-5 Separate Networks for Operation and Diagnostics with 2 DNR Racks and Switches	26
2-6 Typical Configuration for a Single DNR-12-1G with a LAN Switch	29
2-7 Address Ranges to be Scanned	30
2-8 Typical Screen for Analog Input Board	31
2-9 Displaying the Version of Your Firmware	32
2-10 Update Firmware Menu Item	33
2-11 Password Dialog Box	34
2-12 Firmware Update Progress Dialog Box	34
2-13 Physical Dimensions of DNR-12-ENCL Enclosure	35
2-14 System Configuration with LAN Switch	36
Chapter 3 PowerDNA Explorer	42
3-1 PowerDNA Explorer Main Window	43
3-2 Preferences	43
3-3 Address Ranges Dialog Box	44
3-4 Edit Address Ranges Dialog Box	44
3-5 After a Network >>Scan Network	44
3-6 Password Dialog Box for “Store Config” and “Store All Configs”	45
3-7 Password Dialog Box for “Update Firmware .”	45
3-8 Example of a Wiring Diagram Display	46
3-9 Example of the Device Tree	47
3-10 Example of IOM Settings Panel for a DNR-12-1G	47
3-11 Example of I/O Device Settings	48
3-12 Screen from Network >> Read Input Data	49
3-13 Example DIO-405 Inputs	50
3-14 Example DIO-405 Outputs	51
3-15 Example of DIO-403 Inputs	52
3-16 Example of DIO-403 Outputs	52
3-17 Example of DIO-403 Configuration	53
3-18 Example DIO-403 In/Outputs	53
3-19 Example AO-302 Module	54



3-20	Example AI-201 Module.....	55
3-21	Example CT-601 Module	56
3-22	Example Quadrature Controls	57
3-23	Example Bin Counter Controls.....	57
3-24	Example Pulse Width Modulation (PWM) Controls.....	57
3-25	Example Pulse Period Controls	57
3-26	Example of Started Counter.....	58
Chapter 4 The DNR-CPU-1000 Core Module		58
4-1	PowerDNR Core Module (CPU/NIC) – DNR-CPU-1000	59
4-2	FreeScale PowerPC CPU/NIC Controller Architecture.....	60
4-3	CPU / NIC Pinout Diagrams.....	62
Chapter 5 Programming Module-specific Functions		62
5-1	TCP/IP Properties	69
5-2	Core Module Interconnection Diagram	75
Chapter 6 Host / IOM Communication		88
6-1	Communicating with an IOM.....	78
6-2	ACB and Ring Buffer Overall Structure.....	80
6-3	Packet Ring Buffer	81
6-4	DMap Operation.....	84
6-5	Data Field of a RDFIFO Packet Containing Messages.....	88
6-6	Modes of Operation	88
Chapter 7 DaqBIOS Protocol		103
7-1	DaqBIOS Packet Over UDP Packet	93
7-2	DaqBIOS Packet Over Raw Ethernet Packet	93
7-3	Host/IOM Packet Exchange in DaqBIOS Protocol.....	95
Chapter 8 DaqBIOS Engine		107
8-1	User Application/DQE/IOM Interaction.	97
Chapter 9 Real-time Operation with an IOM		110
Appendix A		120
(None)		
Appendix B		133
B-1	Location of Fuse for PL-61x, PL-62x, and PL-63x Boards.....	123
B-2	Location of Fuses for DNR-POWER-DC Board	124
B-3	Location of Fuses for DNA-POWER-1GB Board	124
Index		135



Table of Tables

Chapter 1 DNR-12 RACKtangle™ System	1
(None)	
Chapter 2 Installation and Configuration	19
2-1 Port States	38
2-2 List of Functions and Associated Layers	42
Chapter 3 PowerDNA Explorer	42
(None)	
Chapter 4 The DNR-CPU-1000 Core Module	58
(None)	
Chapter 5 Programming Module-Specific Functions	62
5-1 Memory Map	63
Chapter 6 Host / IOM Communication	88
(None)	
Chapter 7 DaqBIOS Protocol	103
(None)	
Chapter 8 DaqBIOS Engine	107
(None)	
Chapter 9 Real-time Operation with an IOM	102
9-1 RtDMap API Functions	100
9-2 RtVmap API Functions	104
9-3 Output VMap Buffer	107
9-4 Input VMap Buffer	107
Appendix A	120
(None)	
Appendix B	133
B-1 DNA/DNR Replacement Fuses	123
Index	135



Chapter 1 The DNR-12-1G RACKtangle System

This document describes the features, performance specifications, and operating functions of the DNR-12-1G RACKtangle™ data acquisition system. The system is designed for use with an Ethernet Gigabit 1000 Base-T communication network.

1.1 Organization of Manual

This DNR-12-1G User Manual is organized as follows:

- **DNR-12-1G RACKtangle™ System**
This chapter provides an overview of a DNR-12-1G system, component modules, features, accessories, and a list of all items you need for initial operation.
- **Installation and Configuration.**
This chapter summarizes the recommended procedures for installing, configuring, starting up, and troubleshooting a DNR-12-1G system.
- **PowerDNA Explorer for the DNR-12-1G**
This chapter provides a general description of the menus and screens of PowerDNA Explorer when used with a DNR-12-1G system.
- **The DNR-CPU-1000Core Module (CPU/NIC)**
This chapter describes the DNR-CPU-1000 module, which contains a PowerPPC CPU and a GigE Network Interface Module.
- **Programming Board-specific Functions**
This chapter describes tools and facilities used for programming board-specific functions.
- **HOST / IOM Communication**
This chapter describes the various operating modes that may be used in a PowerDNR system for controlling communications between host and IOM.
- **DaqBIOS Protocol**
This chapter describes how the DaqBIOS protocol works.
- **DaqBIOS (DQE) Engine**
This chapter describes the operation and application of the DQE Engine.
- **Real-time Operation with an IOM**
This chapter discusses operation of the DNR-12-1G system under control of a real-time operating system.
- **Appendix A – Configuring Additional Ethernet Cards**
This appendix describes procedures for installing and configuring additional Ethernet cards for us with various popular operating systems.
- **Index**
This is an alphabetical listing of topics covered in the manual, identified by page number.



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 ways to get the job done, or reveal good ideas you might not discover on your own.

NOTE: Notes alert you to important information.



***CAUTION!** advises you of precautions to take to avoid injury, data loss, and damage to your boards or a system crash.*

Text formatted in **bold** typeface generally represents text you should enter verbatim. For instance, it can represent a command, as in the following example: “You can instruct users how to run setup using a command such as **setup.exe**.”



Before plugging any I/O connector into the Cube or Board, be sure to remove power from all field wiring. Failure to do so may cause severe damage to the equipment.

Usage of Terms



Throughout this manual, the term “Cube” refers to either a PowerDNA Cube product or to a PowerDNA RACKtangle™ rack mounted system, whichever is applicable.



1.2 PowerDNR DNR-12-1G System

The UEI PowerDNR DNR-12-1G RACKtangle™ product is a rack-mounted version of the popular PowerDNA Cube Ethernet-based Data Acquisition System. The DNR-12-1G houses a PowerDNA data acquisition system in a rack enclosure accessible from the front of the rack. Multiple DNR-12-1G systems may be mounted in a single rack. All standard PowerDNA I/O boards are also available in PowerDNR versions for use in DNR-12-1G systems.



Figure 1-1. UEI Typical PowerDNR DNR-12 RACKtangle™ System

As illustrated in **Figure 1-4** and **Figure 1-6**, a standard PowerDNR rack system consists of the following modules:

- One or more DNR-12-ENCL rack mounted Enclosures
- DNR-POWER-DC Power Module (one for each enclosure)
- DNR-CPU-1GB Module (Freescale MPC8347 CPU and 1-GB Ethernet 1000 Base-T Network Interface Module — one for each enclosure)
- DNR-BUFFER Board Module (one for each enclosure)
- DNR-IO-FILLER panels (one for each unused I/O slot)
- DNR-PSU-24-100 100-Watt, 120/230 VAC to +24VDC External Power Supply (one for each enclosure) with cable and Molex connector for plug-in to the DNR-POWER-DC Module front panel.

To configure a complete data acquisition system, insert up to 12 DNR I/O modules into each PowerDNR rack enclosure, which may be specified in any combination of the following types:

- DNR-AI-201, -202, 205, 207, -208, -211, -225
- DNR-AO-308, -308-350, -308-353, -308-420, -332,
- DNR-DIO-401, -402, -403, 404, -405, 406, -416, -432, -433, -448
- DNR-CT-601, DNR-QUAD-604
- DNR-SL-501, DNR-CAN-503
- DNR-429-566, DNR-429-512
- DNR-GPS
- Any future additions to the PowerDNR I/O module product line

Note: Refer to www.ueidaq.com for a description of each I/O module.



All standard PowerDNA accessories are also available for use in a PowerDNR rack-mount system.

1.3 Specifications

Figure 1-2 lists the technical specifications of the PowerDNR system.

Standard Interfaces	
To Host Computer	Two independent 1000Base-T Gigabit Ethernet ports via RJ-45 connector
Distance from host	100 meters, max
Other Interfaces	One USB 2.0 controller port, One USB 2.0 slave port.
Config/General	RS-232, 9-pin "D"
Sync	Custom cable to sync multiple racks
I/O Slots Available	
DNR-12-1G	12 slots
Data transfer and communications rates	
Ethernet data transfer rate	20 megabytes per second
Analog data transfer rate	up to 6 megasample per sec (16-bit samples)
DMA I/O mode	update 1000 I/O channels (analog and/or digital) in less than 1 millisecond, guaranteed
Processor	
CPU	Freescape 8347, 400 MHz, 32-bit
Memory	128 MB (not including on-board Flash, upgradable to 1 Gb)
Status LEDs	Power supplies within spec, One second system heart-beat, Attention, Read/Write, Power, Communications Active
Environmental	
Temp (operating)	Tested to -40 °C to 70 °C
Temp (storage)	-40 °C to 100 °C
Humidity	0 to 95%, non-condensing
Vibration	
(IEC 60068-2-64)	10-500 Hz, 3 g (rms), Broad-band random
(IEC 60068-2-6)	10-500 Hz, 3 g, Sinusoidal
Shock	
(IEC 60068-2-27)	50 g, 3 ms half sine, 18 shocks at 6 orientations; 50 g, 11 ms half sine, 18 shocks at 6 orientations
Physical Dimensions	
DNR-12 series	5.25" x 6.2" x 17.5" (3U in a 19" rack)
Power Requirements	
Voltage	9 - 36 VDC (AC adaptor included)
Fuse	Internal 10 A
Power Dissipation	13 W at 24 VDC (not including I/O boards)
Power Monitoring	
I/O board power	All internal power supplies monitored to $\pm 1\%$ accuracy. All PS voltages may be read by host. LED annunciators indicate out of range
Input current	Monitored by host, LED indicates overcurrent
Input voltage	Monitored by host, LED indicates out of range

Figure 1-2. Technical Specifications

1.4 Key Features The following table is a list of key features of a DNR-12-1G PowerDNR system.

Easy to Configure and Deploy ● Over 25 different I/O boards available ● Over 5 quadrillion possible configurations ● Built-in signal conditioning ● Gigabit Ethernet based ● Bracket kit for mounting to wall or in 19" racks ● Industrial quality rubber feet for solid table-top mounting ● Passive backplane ensures extremely low MTTR ● Standard "Off-the-shelf" products and delivery True Real-time Performance ● 1 msec updates guaranteed with 1000 I/O ● Up to 6 million samples per second ● Use QNX, RTX, RT Linux, RTAI Linux, LabVIEW RT Flexible Connectivity ● 1000Base-T with Cat-5 cable ● 10/100Base-FX Fiber interface available (see DNR-FPPC family) ● Supports WIFI / GSM / Cell networks ● Built-in USB 2.0 slave and controller ports Compact Size: ● 5.25" x 6.2" x 17.5" ● 300 analog inputs per rack ● 384 analog outputs per rack ● 576 digital I/O bits per rack ● 96 counter/quadrature channels per rack ● 144 ARINC-429 channels per rack ● 48 RS-232/422/485 ports per rack Low Power: ● Less than 50 watts per typical rack (including I/O)* ● AC, 9-36 VDC or battery powered. Stand alone and Data Logger Modes ● Upgradable to UEILogger 1200R ● Upgradable to UEIPAC 1200R ● Upgradable to UEIModbus 1200R Rugged and Industrial: ● Solid Aluminum construction ● Operation tested from -40°C to +70°C ● Vibration tested to TBD g, (operating) ● Shock tested to TBD g (operating) ● All I/O isolated from rack and host PC. ● Operation to TBD feet Outstanding Software Support ● Windows, Linux, RT Linux, Windows RT, RTX and QNX operating systems ● VB, VB .NET, C, C#, C++, J# ● MATLAB, LabVIEW, LabVIEW RT, DASyLab, OPC, ActiveX *Varies with layer types, Refer to layer datasheets.
--

Figure 1-3. PowerDNR Product Features



1.5 DNR-12-1G PowerDNR System Enclosure

Because a major design goal of the DNR product line is to allow easy configuration and updates in the field, each DNR-12-1G enclosure contains a DNR-POWER-DC Power Module with status indicating LEDs and a local ON/OFF switch — and each I/O module is provided with a mechanical lever (see **Figure 1-12**) for quick ejection and insertion of the board. In addition, each DNR-12-1G system enclosure also contains its own GigE CPU and two Network Interface Control (NIC) modules, one for controlling up to 12 I/O modules mounted in the enclosure, and another for diagnostics functions. The module-specific I/O boards are functionally identical to the corresponding modules for the PowerDNA Cube. The only differences between the two types relate to the mounting arrangements.

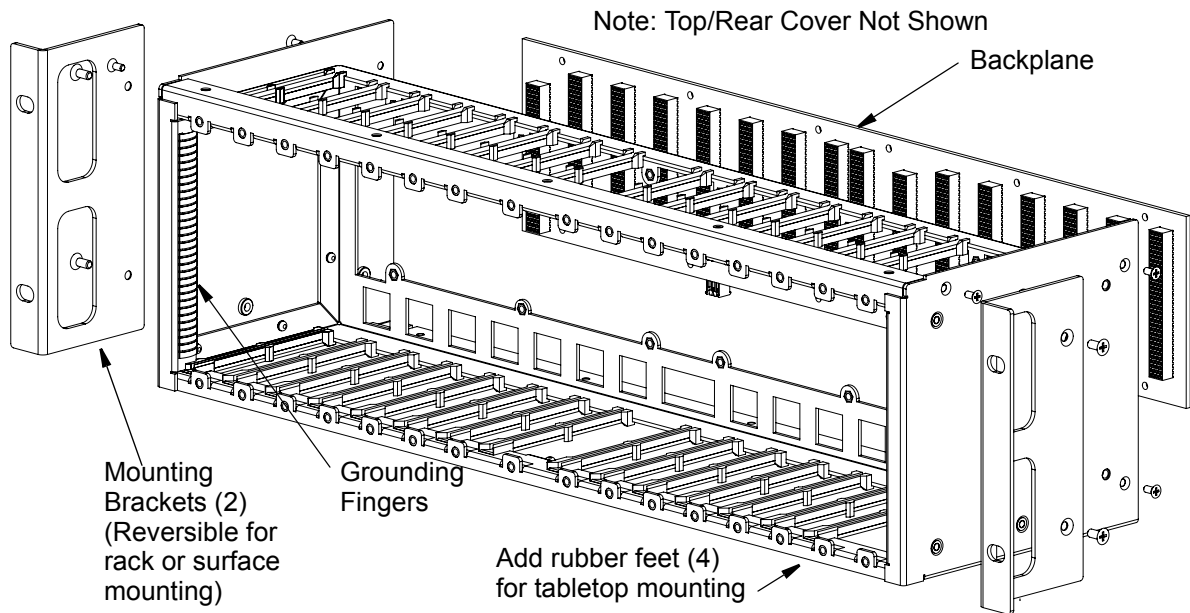


Figure 1-4. Typical PowerDNR DNR-12-ENCL Enclosure (Exploded View)

As shown in **Figure 1-4** and **Figure 1-6**, the PowerDNR DNR-12 enclosure is designed to house the following items:

- One isolated DNR-POWER-DC DC/DC Power Module/Power Monitor with status indicating LEDs, a local on/off switch, and 4-pin Molex Power-In connector
- One DNR-CPU-1000 dual-slot CPU/NIC module with indicating LEDs, two Ethernet connectors (Main and Diagnostic Ports), sync connector, reset pushbutton, SD card slot, USB controller/slave ports (future use), and a DB-9 connector for a serial port
- One DNR-BUFFER Board, for buffering address/control/clock lines (for future use — not currently addressable)
- Up to 12 PowerDNR front pull-out I/O modules (boards) functionally identical to PowerDNA I/O boards but designed for mounting in a DNR rack enclosure
- One DNR-BP-12 Backplane with two temperature sensors
- DNR-IO-FILLER blank filler panels for all unused slots



- Four 8-volt cooling fans mounted on the rear of the enclosure

NOTE: Note that the rightmost module (Module 11) is 2-slots wide (to accommodate future designs and/or custom modules).

Also note that the DNR-12 enclosure has reversible mounting flanges designed for rack or surface mounting. Rubber feet are supplied for desktop or tabletop mounting.

The enclosure is a rigid mechanical structure with complete EMI shielding. Unused slots are filled with blank brackets and filler panels. The DC/DC power module provides output voltages of 24, 3.3, 2.5, 1.5, and 1.2 VDC for the logic/CPU and 8 VDC to power the cooling fans.

1.5.1 Cooling Air Flow

As shown in **Figure 1-5**, cooling is drawn into the rear of the enclosure, routed forward over the electronic circuit boards, up to the top of the enclosure, and then out the top rear of the enclosure. The system is designed to maintain positive pressure cooling within the enclosure at all times.

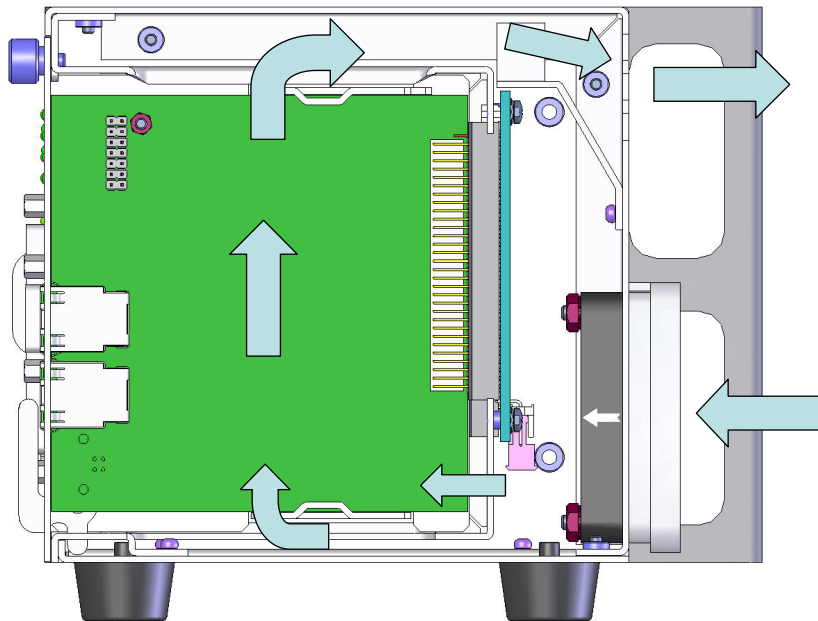


Figure 1-5. DNR-12 Air Flow



1.5.2 DNR Power, CPU/NIC, and I/O Modules

This section describes the basic modules included in every RACKtangle system, the CPU/NIC module, the DC/DC power module, buffer module, and I/O modules.

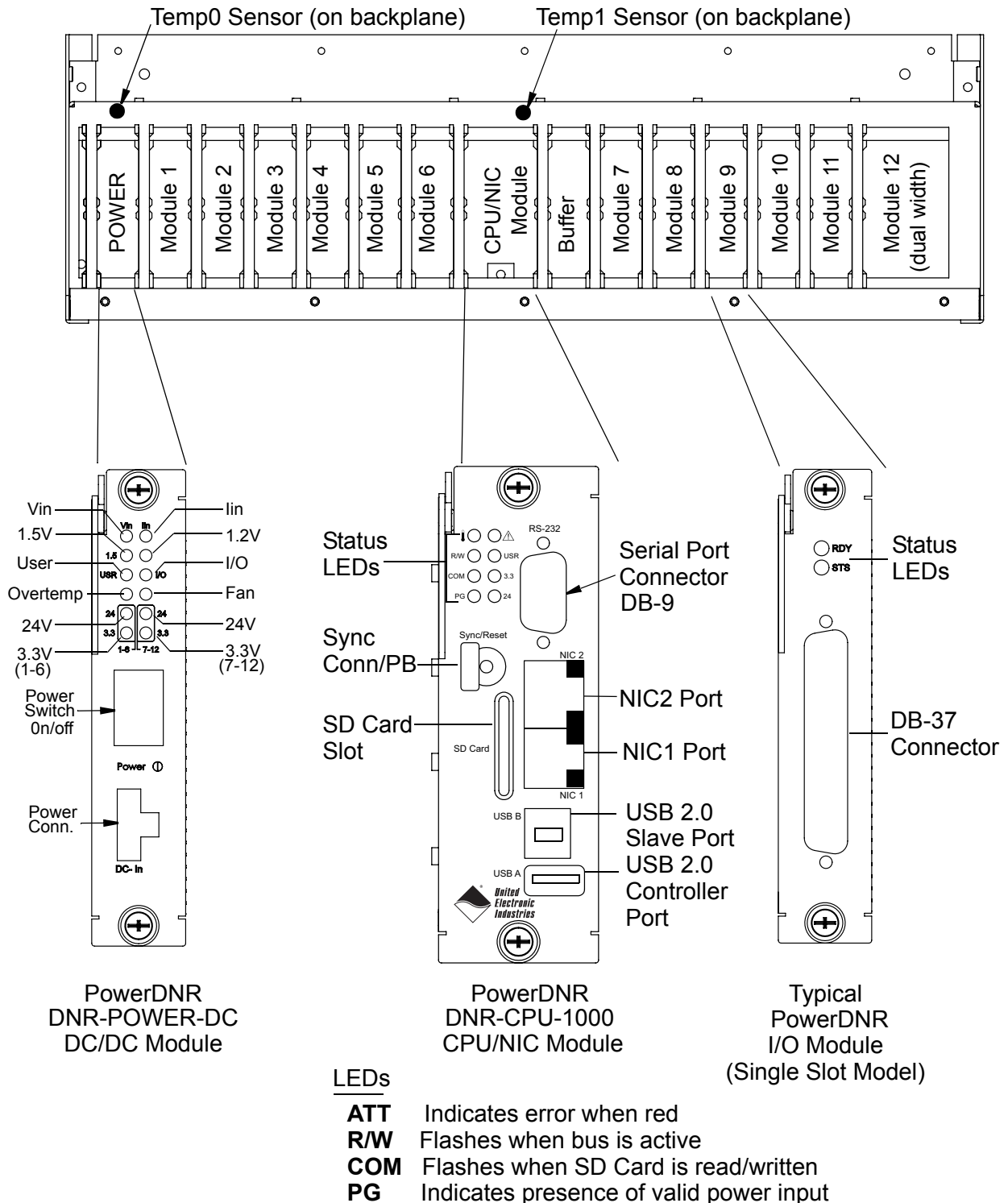


Figure 1-6. PowerDNR System Front Panel Arrangement



Figure 1-7, Figure 1-8, and Figure 1-9 describe the conditions indicated by the LEDs on the front of each module in the rack.

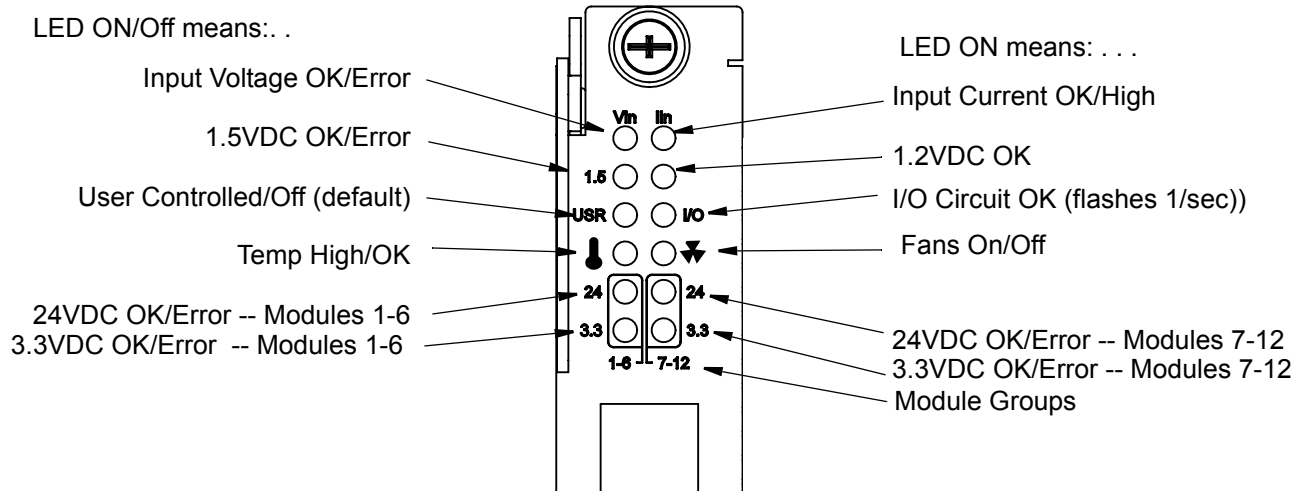


Figure 1-7. DC Power Module LEDs

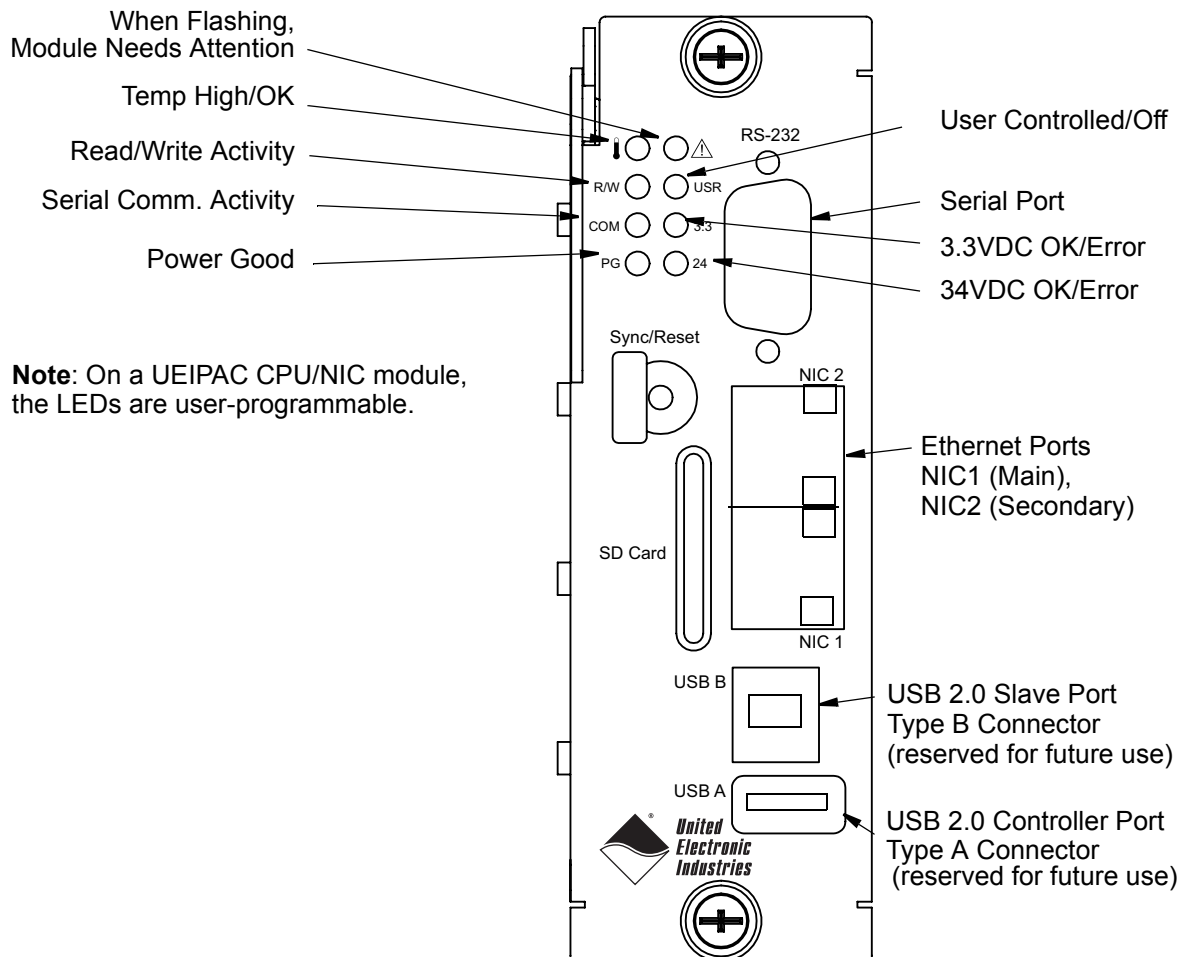


Figure 1-8. DNR-CPU-1000 Module LEDs



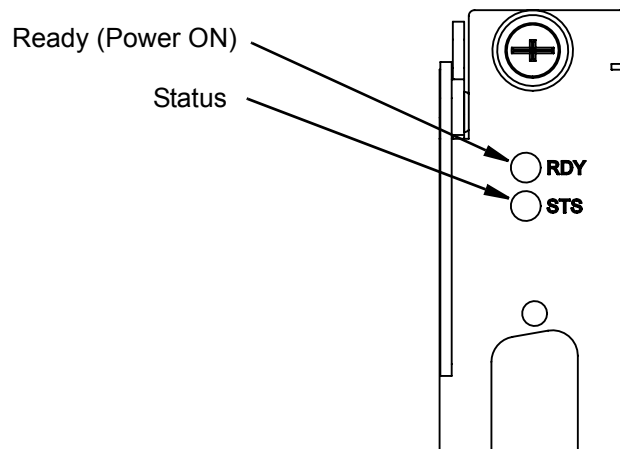


Figure 1-9. Typical I/O Module LEDs

Two sensors mounted on the backplane over the Power Module and over the CPU board monitor internal temperatures continuously, turning fans on if the internal temperature exceeds 45°C, off if it falls below 45°C, and shutting down power if a high limit is exceeded.

All UEI PowerDNA modules are available in both PowerDNA and Power DNR package designs.

A feature of the design is that the address of a module is determined by the position of the module within the enclosure, numbered from left to right. A typical module address is:

0xA00nxxxx

where **A00** is the BASE address

n is the module position number starting from 0 at the left

xxxx is the address of the module

With this addressing method, the address of a given I/O board (module) automatically changes if you move it from one position to another within the enclosure.

The slots or module positions are numbered as follows:

Physical Position (L-R)	Position Number	Module Description
1	0xC	POWER
2	0x0	Module1
3	0x1	Module2
4	0x2	Module3
5	0x3	Module4
6	0x4	Module5
7	0x5	Module6
8	0xE	CPU/NIC
9	0xD	BUFFER
10	0x6	Module7
11	0x7	Module8
12	0x8	Module9
13	0x9	Module10
14	0xA	Module11
15	0xB	Module12



1.6 DNR-POWER-DC Module

The DNR-POWER-DC Module is a dedicated DC/DC source and control module available only for use with the PowerDNR rack enclosure. It is always mounted in the leftmost slot of the DNR chassis and is recognized on the PowerDNR bus with an ID of 0x020 at address 0xA00C0000.

The non-isolated side (NIS) logic complies with full common logic interface (CLI) implementation. The key features of the DNR-POWER Module are:

- Input power — 9-36 VDC 80W maximum, protected by resettable fuses and EMI chokes
- Power supply on/off switch (with guard)
- Output power sources (all with greater than 90% efficiency)
 - 24V, 1A (24W)
 - 3.3V, 5A (16.5W, including the 2.5V derived voltage)
 - 2.5V, 3A (derived from 3.3V source)
 - 1.5V, 5A, (7.5W, including the 1.2V derived voltage)
 - 8V, 0.5A (4W for fans)
- DC/DC for 24V, 3.3V, and 1.5V are synchronized from the single spread-spectrum clock source in the CPU/NIC Module for lower EMI noise level
- Fan control (Forced ON) and status ON/OFF
- Monitoring and LED indicators (1% accuracy, 0.25Hz update rate) for:
 - All output voltages
 - Input current for the 9-36VDC for the DNR Enclosure
 - All voltages from the NIC Module (24V, 3.3V, 2.5V)
 - Temperature of the DNR backplane (2 sensors)
- Onboard FPGA logic chip is CYCLONE EP1C3/C6T144
- TI MSP4300 microcontroller used for logic reprogramming
- Input-Output connector is a 128-pin component that provides 9-36VDC for all modules from an external power source



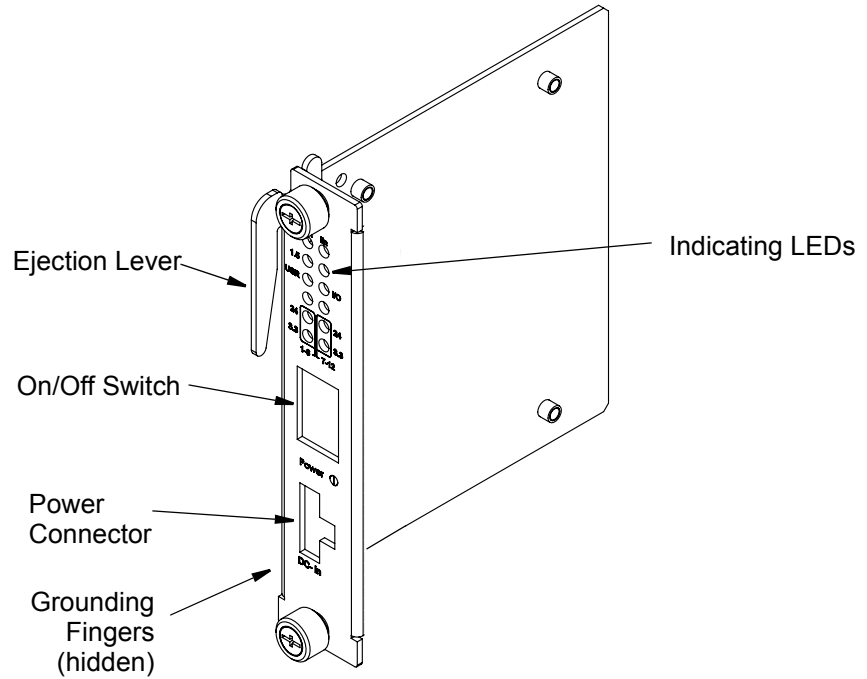


Figure 1-10. DNR-POWER-DC Module

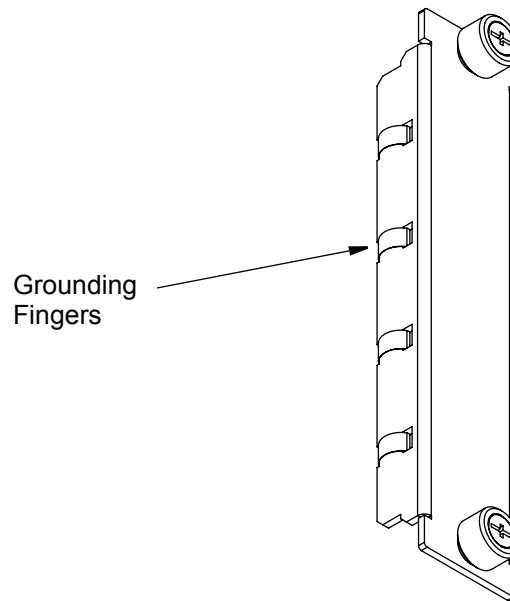


Figure 1-11. DNR Filler Panel for empty slots

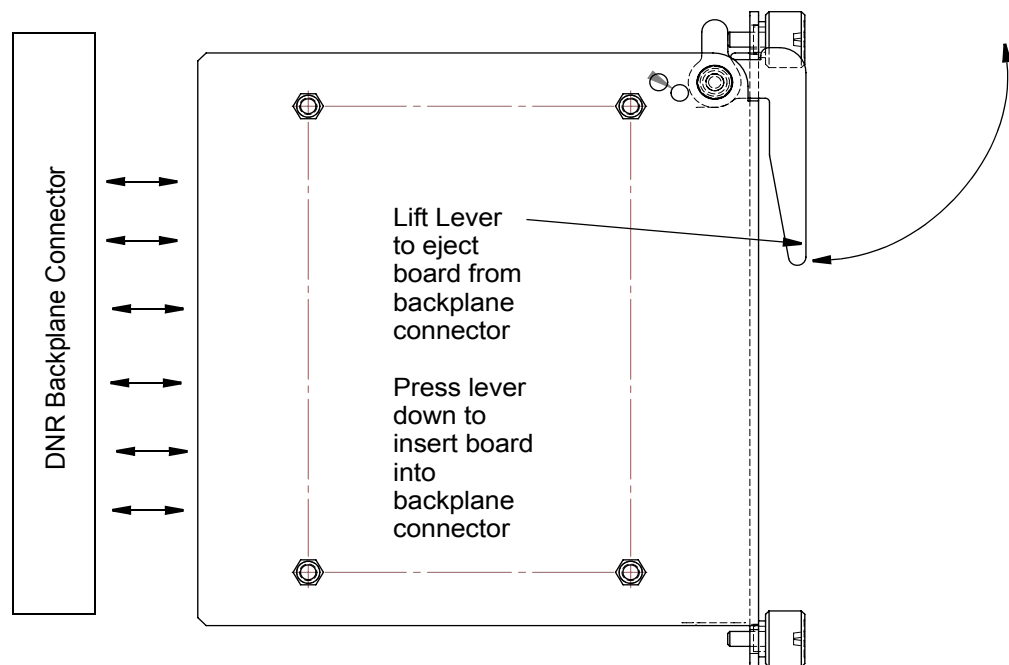


Figure 1-12. DNR Ejection/Insertion Lever Operation



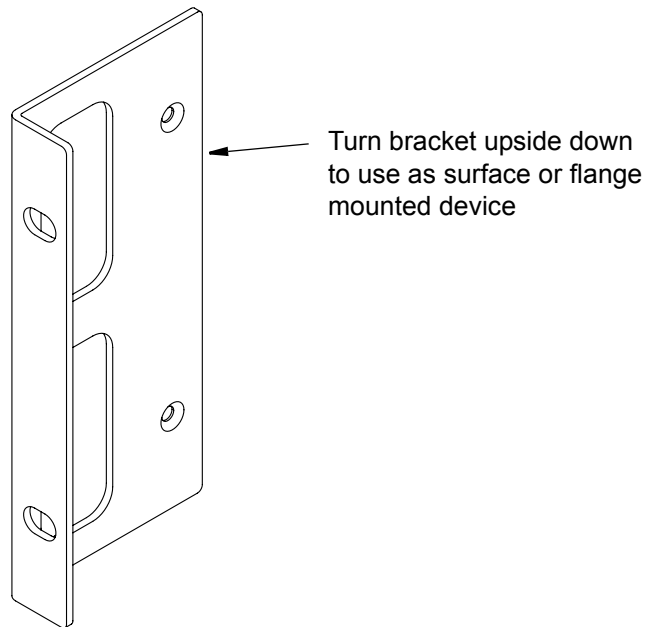


Figure 1-13. DNR-BRACKET Reversible Mounting Bracket



A functional block diagram of the DNR-POWER-DC Module is shown in **Figure 1-14** below.

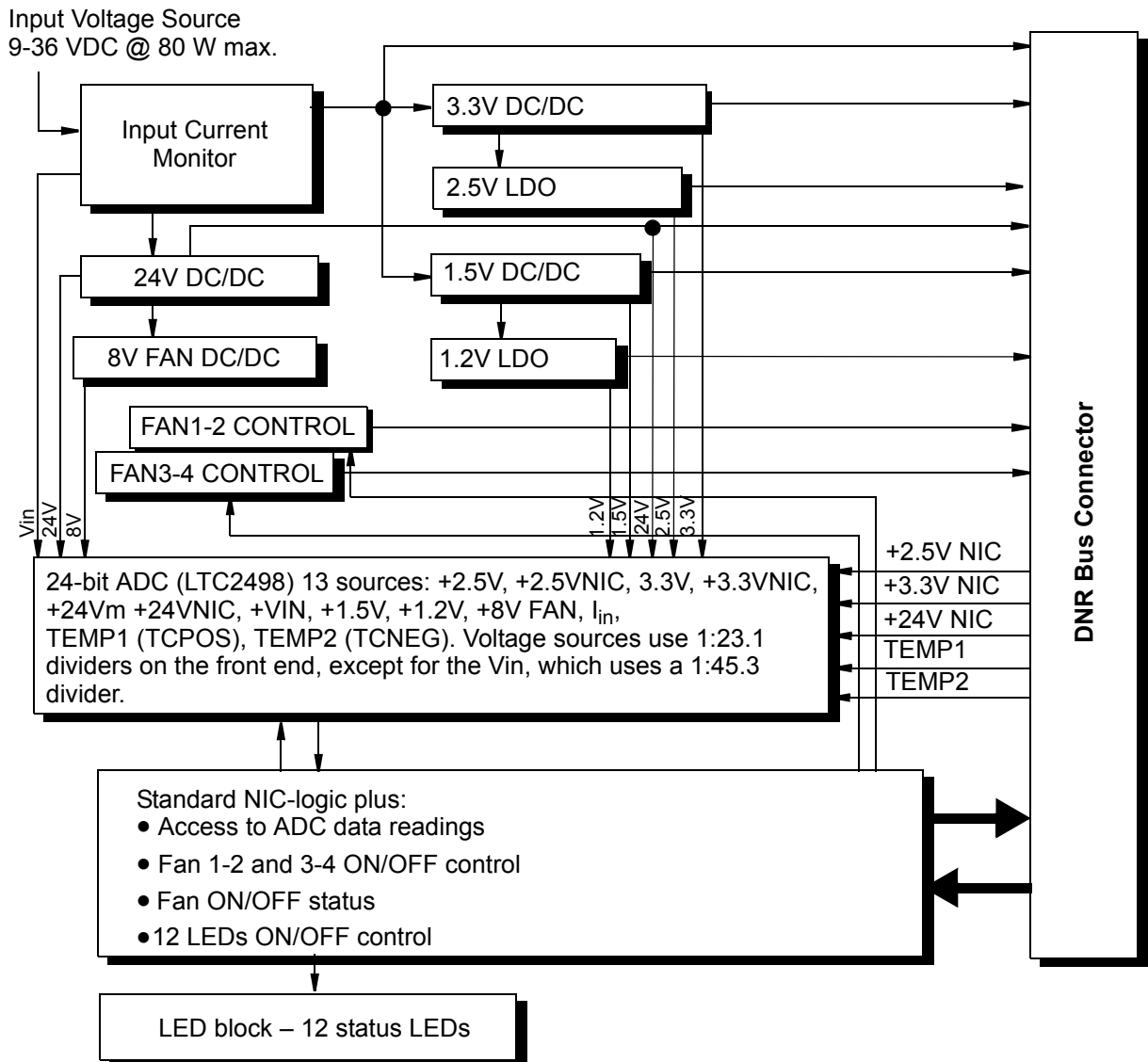


Figure 1-14. Functional Block Diagram of DNR-POWER Module

As shown in **Figure 1-14**, the DNR-POWER-DC Module operates as follows:

A 9-36VDC voltage input (Vin) from an external source is connected to the board through a resettable fuse. The board monitors the input current and passes Vin to the DNR bus as Vout. Vout also is connected to DC/DC converters that produce 24 VDC, 3.3VDC and 1.5VDC output voltages, which are also placed on the DNR bus. Both 3.3 and 1.5VDC voltages are connected to low dropout regulators that, in turn, generate the 2.5VDC and 1.2VDC output voltages on the bus. The 24VDC source is fed to a low dropout regulator that produces 8VDC to drive the cooling fans (through fan controller chips).

The input current and all output voltages, including the +2.5, +3.3, and +24VDC from the NIC module, plus signals from the two temperature sensors mounted within the enclosure, are input to a 24-bit delta-sigma A/D converter. Except for Vin, the voltage sources use 1:23.1 dividers on the front end. Vin uses a 1:45.3 divider.

Figure 1-15 shows the interaction of modules within a DNR-12-ENCL Enclosure when the DNR-BUFFER module is used.

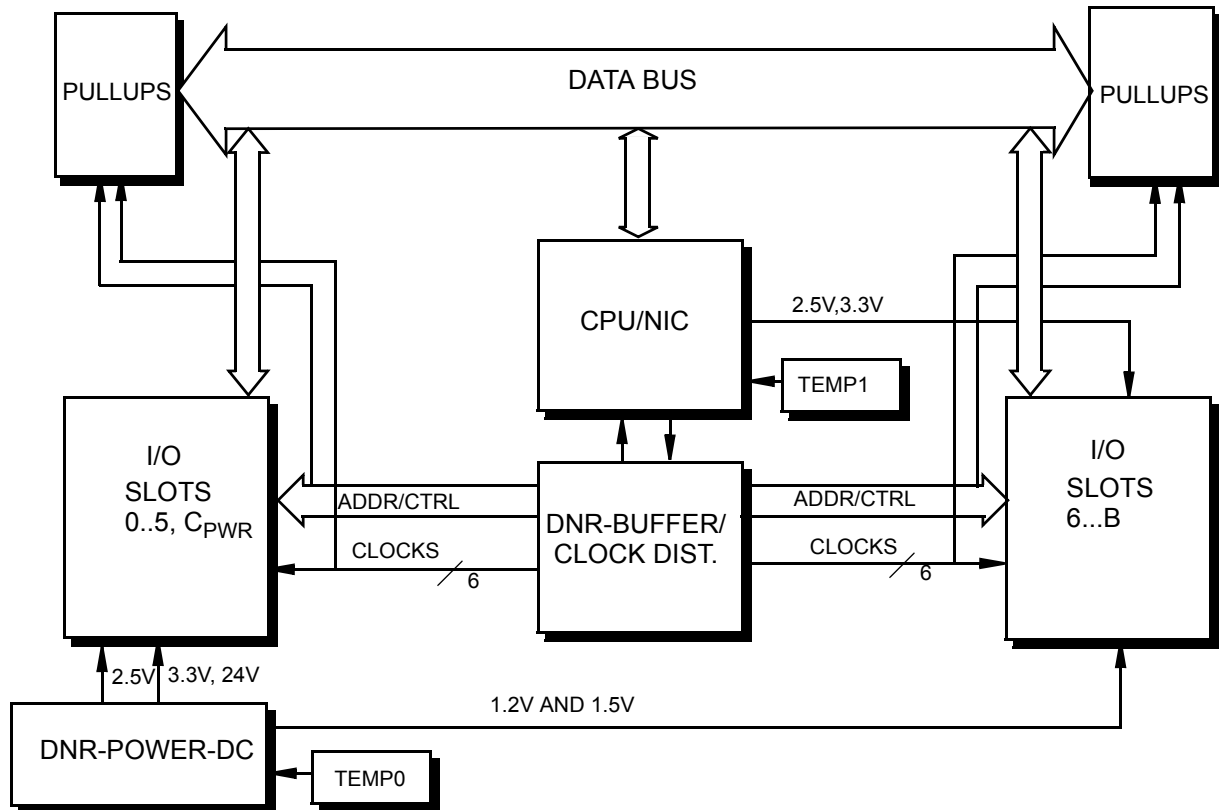


Figure 1-15. Functional Block Diagram of DNR-12-ENCL

As shown above, the I/O slots are divided into two groups: 0 to 5 and 6 to 0xB. 0xC for the DC Power Module is included with the 0 to 5 group. The DNR-BUFFER board is located at the center of the enclosure, which is also at the center point of the ADDR/CTRL bus. The DNR-CPU-1000 module is also located at the center of the enclosure and the center of the data bus to minimize bus delays. The CPU addresses I/O modules and transmits clock ticks through the Buffer Board, which controls the Addr/Ctrl and clock lines to the modules.

Temperature sensors monitor temperatures within the enclosure above the DNR-POWER-DC module and the DNR-CPU module.

1.7 DNR-CPU/NIC Module

The DNR-CPU-1000 Module contains a PowerPC 8347 CPU and associated Network Interface Control (NIC) logic that controls all Ethernet communication functions. The DNR-CPU-1000 has a dual 1-GB Ethernet module.



1.8 DNR-Buffer Module

The DNR-BUFFER Module provides buffering between the CPU and I/O module address/control/clock lines, which functions as described in **Figure 1-15**. Although the module may not always be required, it is included to provide an extra margin of safety against loss of data.

1.9 DNR-I/O-Modules

All standard PowerDNA I/O modules are also available as PowerDNR modules. A typical PowerDNR module has functions that are functionally identical to its corresponding PowerDNA version. The only difference between them is the physical mounting arrangement. PowerDNR modules are designed for insertion into the DNR-12-ENCL enclosure; PowerDNA modules can be inserted only into a PowerDNA Cube.

Therefore, for detailed electrical specifications and user instructions for a specific DNR I/O board, refer to the datasheets and User Manuals for the equivalent PowerDNA I/O module. These documents are available for examination and download from the UEI website at www.ueidaq.com.



Chapter 2 Installation and Configuration

Installation consists of:

- DNR-12-1G hardware setup
- PowerDNA/PowerDNR software package installation
- Configuration

2.1 Initial Installation Guide

This section describes the procedure recommended for performing an initial hardware and software setup when you first receive a DNR-12-1G system.

2.1.1 Inspect Package

Inspect the contents of the shipping package. With a standard DNR-12-1G system, you should find:

- A DNR-12-ENCL enclosure, preinstalled with a DNR-12-1000 CPU module, DNR-POWER-DC module, DNR-BUFFER module, blank filler panels, plus your selection of I/O Boards.
- A DNA-PSU-24-100 100-watt universal powerline brick that plugs into an AC outlet and provides 24V dc output. The supply comes with a power cord for the mains and an adapter cable ending in a Molex connector for plugging into the DNR-POWER-DC Module.
- DB-9 serial cable for initial hardware configuration and firmware downloading.
- CD-ROM with support software

NOTE: Depending on your application, you may also need to provide the following items not normally included with your order (see Figures 2-3 to 2-6 on pages 25 to 29):

- CAT5e cables between host and DNR system NIC1 primary port, and/or between host and an Ethernet switch, and/or between switch and DNR primary port
- CAT5e cables between host and DNR system secondary (diagnostic) NIC2 port, between host and an Ethernet switch, and/or between switch and DNR secondary (diagnostic) port

2.1.2 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 or DNR-12-1G support software is online at 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 one installer that combines the UEI low-level driver and UEIDAQ Framework.



The installer automatically searches for third-party IDE and testing suites, and adds them as tools to the suites found. Be sure to install third-party applications (such as LabVIEW, MATLAB, or MsVS2003) **before** installing the PowerDNA Software Suite.

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, double-click on the filename to run the executable.

- b. Choose the PowerDNA Software Suite option.
- c. Unless you are an expert user and have specific requirements, 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 necessary packages are installed.

-- Companion Documentation:

Quick Start Guide, Configuration and Core Module,
I/O Board Manuals, Low-Level Programming Guide

-- SDK: includes/lib for C/Java, examples, and Sun's JRE;
(The SDK is not the UeiDaq Framework).

-- PowerDNA Apps: PowerDNA Explorer, MTTTY

-- PowerDNA Components (incl. DLL files)

-- PowerDNA Firmware

- d. Click **Next** to continue through the dialogs.
- e. Click **Finish** to complete the 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:

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

Follow the instructions in the readme.txt file contained therein.

2.2 Initial Boot-up

Perform an initial boot in preparation for configuring the network, using the following procedure:

- STEP 1:** Familiarize yourself with the DNR-12 system front-panel layout. Note that all connections are made on the 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 RS-232 port on the front panel of the DNR-CPU-1000 Module.
- Run a terminal-emulation program (MTTTY) on the PC. Any terminal-emulation program, except HyperTerminal, may be used (MTTTY, Minicom, TeraTerm, etc.).
 - Verify that COM parameters are set at: 57600 baud, 8 bits, no parity, 1 stop bit.
 - Click **Connect** in MTTTY, or use the commands on one of the other terminal-emulation programs to establish communication with the DNR-12-1G system.
- STEP 3:** Connect power to the system (9-36V DC) by plugging the Molex-type power connector from the power supply into the mating connector on the DNR-POWER-DC module. The power source may be the bundled DNA-PSU-24-100 100 watt powerbrick or a user-supplied source. Note that the DNA-PSU-24-100 plugs into a 100 - 240V, 47- 63 Hz outlet and outputs up to 4.17A at 24 VDC.
- STEP 4:** Turn on the ON/OFF power switch on the DNR-POWER-DC Power Module front panel.



STEP 5: As soon as the system powers up, it runs through self-diagnostic mode and generates output on the terminal program. A typical readout might be as shown in **Figure 2-1**:

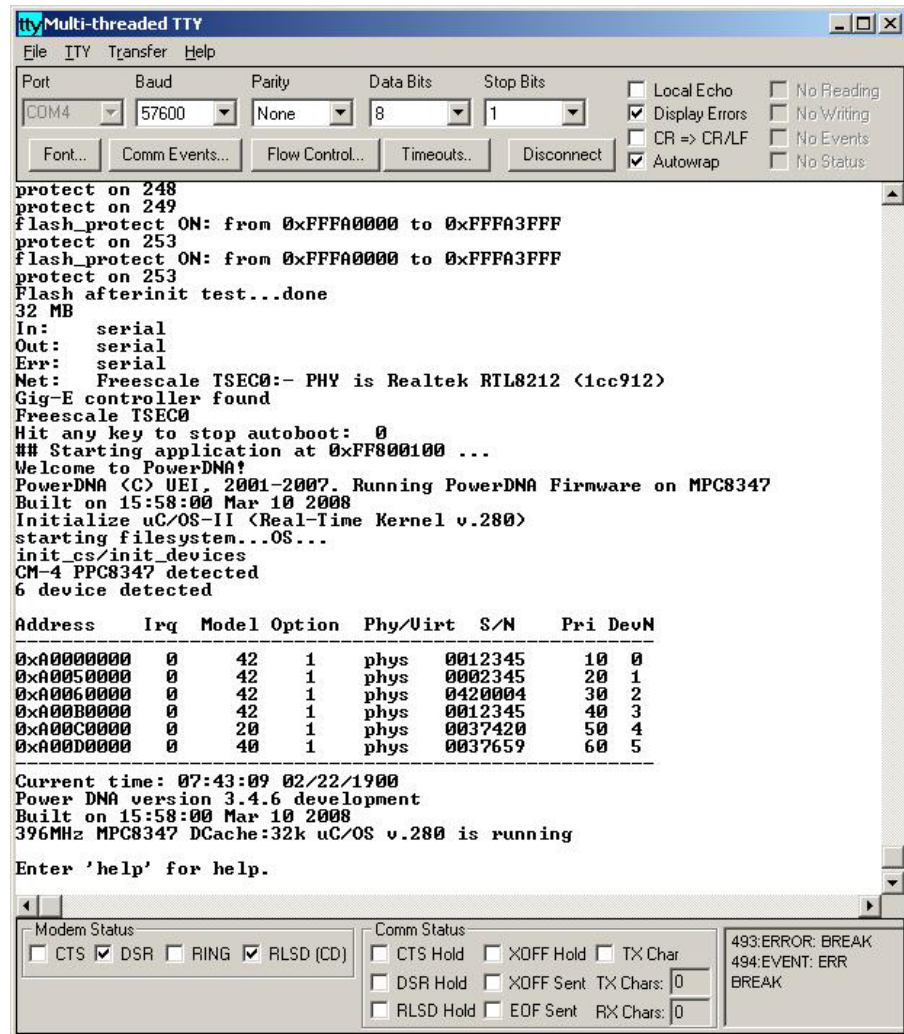


Figure 2-1. Typical MTTY Screen

The boot process displays the model, serial number, and slot positions of boards in the rack enclosure. Type `show <CR>` to display information on system configuration, as illustrated in **Figure 2-2**.



```
DQ>show
      name: "IOM-37710"
      model: 0x3012
      serial: 0037710
      fwct: 1.2.0.0
      mac: 00:0C:94:00:93:4E
      srv: 192.168.100.2
      ip: 192.168.100.10
      gateway: 192.168.100.1
      netmask: 255.255.255.0
      mac2: 00:0C:94:F0:93:4E
      srv2: 192.168.100.102
      ip2: 192.168.100.110
      gateway2: 192.168.100.1
      netmask2: 255.255.255.0
      udp: 6334
      license: ""
      Manufactured 1/31/2008
      Calibrated 1/31/2008
DQ> █
```

Figure 2-2. Show System Configuration

All parameters can be changed; most notably, the IP address, gateway, and subnet mask (netmask) configured for this system.

2.2.1 IP Addresses on the IOM

The DNR-12-1G ships with preconfigured factory default IP addresses for NIC1 and NIC2 in nonvolatile memory (usually 192.168.100.2 for NIC1 and 192.168.100.102 for NIC2). This is a static IP address; the system never retrieves its IP address from a DHCP server. This section describes why and how to change the default IP addresses.

Should you change the IP Address?

Yes, if you plan to use the DNR-12-1G on a LAN in which.

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

Alternatively, if you plan to use the system for high-speed measurements where high reliability is necessary – a direct connection between the host PC and a NIC¹ is recommended. For a direct connection, refer to “Improving Network Performance” on page 25.

How to change the IP address of the primary Ethernet port.

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

1. NIC - Network Interface Controller; a commercially available Ethernet (i.e. IEEE 802.3-2005) adapter.



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

DQ> set ip 192.168.0.65	// Sets this system IP to 192.168.0.65
Enter user password > powerdna	// The default password is "powerdna"
DQ> store	// Saves the newly changed configuration
DQ> reset	// Reboots the system for the new IP to
	//take effect

In this manner, you can set any parameters listed with the "show" command.

Connect the DNR-12-1G system to your switch with a CAT5e cable.

If you can establish communication with a DNR-CPU-1000, 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. You then enter the new IP address and hit <Return>. This action downloads the new IP address into the system's non-volatile memory. You might also need to change the gateway and network mask to match settings on your LAN.



How to change the IP address of the secondary (diagnostic) Ethernet port.

To change the IP address of the secondary port (NIC2), use the terminal program as with the primary port, but instead use the command:

```
set ip2 aaa.bbb.ccc.ddd
```

where aaa.bbb.ccc.ddd is the new IP address for the secondary port.

Then proceed the same as with the primary port.

2.3 Improving Network Performance

To improve DNR-12-1G network performance, we recommend that instead of connecting to a company-wide network, you use separate commercially available network interface controller (NIC) cards and, where possible, set up a single dedicated mini-network for DNR-12-1G racks for both operation and diagnostics, as shown in **Figure 2-4**. As an alternative, you can configure two separate networks, one for operation and one for diagnostic purposes, as shown in **Figure 2-5**. If you do not need to connect to a company LAN and have only a single DNR-12-1G in your system, you can connect it directly to your host as shown in **Figure 2-3**.

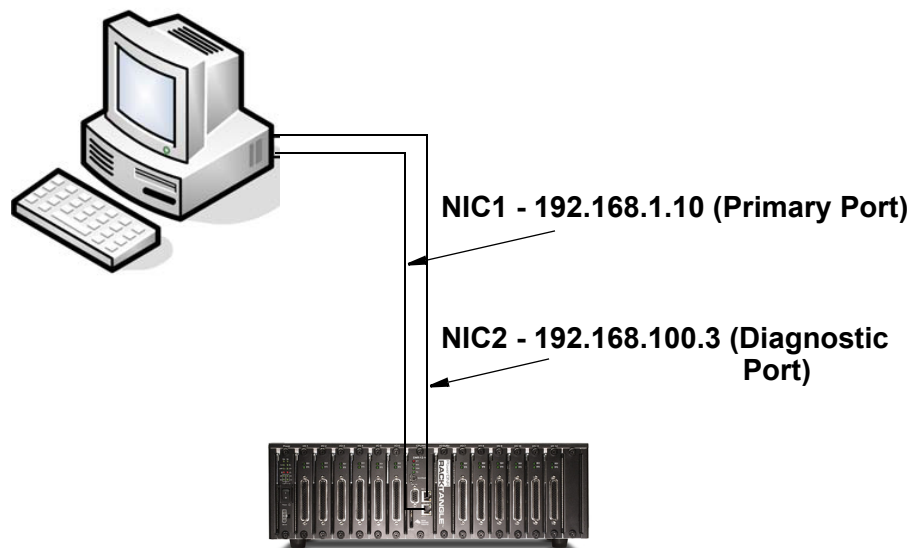


Figure 2-3. Single DNR-12-1G Direct-Connected to Host without LAN Switch



Figure 2-4 shows a two-rack single network system with a LAN switch that performs both data acquisition and diagnostic functions..

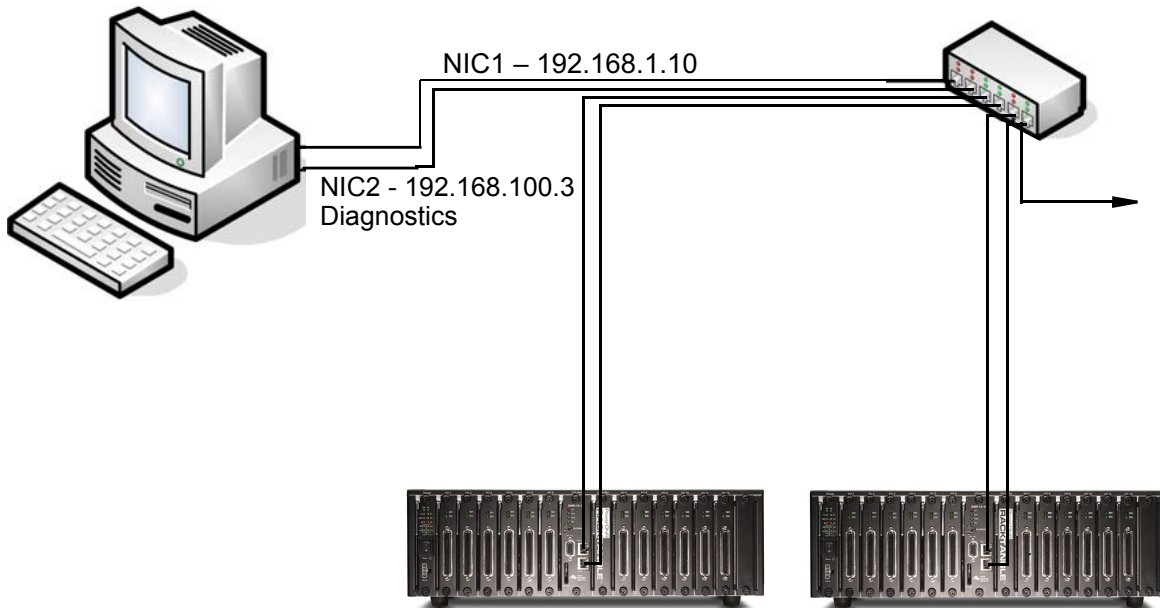


Figure 2-4. Single Network for Both Operation and Diagnostics Using Two DNR Racks and LAN Switch

Figure 2-5 shows a two-rack dual network system with two LAN switches that performs both data acquisition and diagnostic functions.

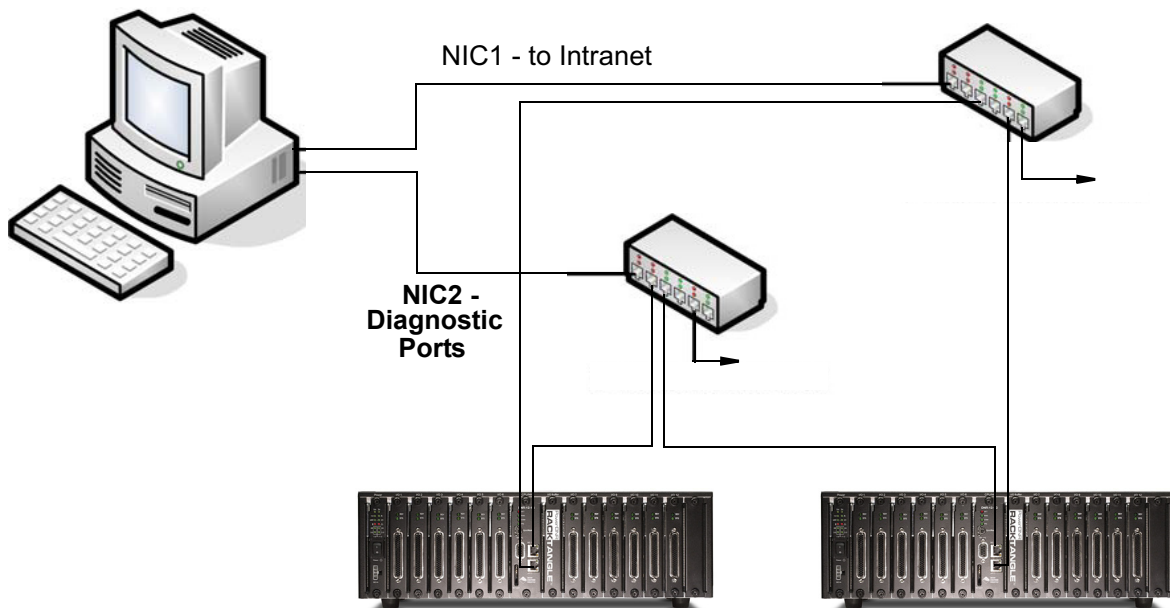


Figure 2-5. Separate Networks for Operation and Diagnostics Using Two DNR Racks and Two LAN Switches



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.

In the following example, the subnet range 192.168.1.0 — 192.168.1.255 is 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 lies within 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 to 192.168.2.255 is available and is 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 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.

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.

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 4: Set up the DNR-12-1G system 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 see the DQ command prompt:

```
DQ> set ip 192.168.100.2 // Sets this Cube's IP address to  
                        // 192.168.100.2  
DQ> set gateway        // Sets this Gateway to  
192.168.100.3           // 192.168.100.3  
DQ> set netmask         // Sets the subnet mask to 255.255.255.0  
255.255.255.0  
DQ> store               // Saves the newly changed configuration  
DQ> reset               // Reboots the system for the new IP to take  
                        // effect.
```

STEP 5: Connect the DNR-12-1G to your PC's second NIC, using a CAT5 cable. The green LEDs should light up.

STEP 6: Ping the system 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 would indicate an error.

STEP 7: The system should now be configured as shown in **Figure 2-6**.

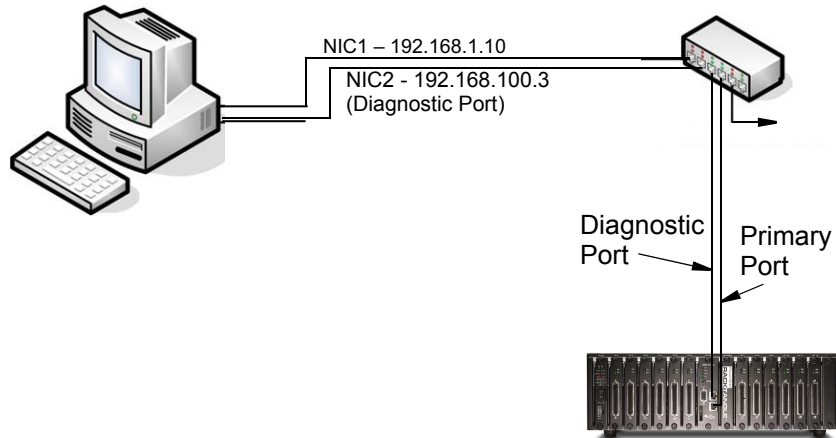


Figure 2-6. Typical Configuration for a Single DNR-12-1G with a LAN Switch

STEP 8: You may now use PowerDNA Explorer to access the system. See Chapter 3.

2.3.1 Troubleshooting

The following checklist may assist you in troubleshooting a system.

- ☒ The PG (Power Good) LED is on: the 9-36V DC power supply is plugged into the DNR-POWER-DC Power Module panel.
- ☒ The green lights on NIC ports are blinking: the CAT5e cables are connected.
- ☒ Use the command prompt to ping <system IP> (e.g. ping 192.168.100.2)
 - a. Disable (temporarily) the firewall on the secondary NIC.
 - b. Check the secondary NIC's network settings.
 - c. Check the system's network settings.
 - Use MTTTY and click 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 click Connect and press [Enter] .
- ☒ Reboot the DNR-12-1G system. The start-up screen should display upon restart.
- ☒ If all else fails, contact UEI support at: support@ueidaq.com.
- ☒ Type “show” to verify the 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.

2.4 PowerDNA Explorer Quick-Start

PowerDNA Explorer does just what its name implies: it “explores” the LAN, looking for connected PowerDNA Cubes and/or DNR-12-1G systems. Chapter 3 covers the PowerDNA Explorer in detail. This section only provides a quick-start guide.

The PowerDNA Explorer identifies DNR-12-1G systems (or Cubes) on a selected network – the discovered systems are listed on the left-hand pane of the display. Select a specific system to display pertinent hardware and firmware information. Select a board of a specific system to manipulate its inputs or outputs. In brief, this useful tool lets you verify that the system is communicating with the host and that the I/O Boards are functioning properly.

To scan the network for DNR-12-1G systems or Cubes, provide a set of addresses to scan. Do the following:

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

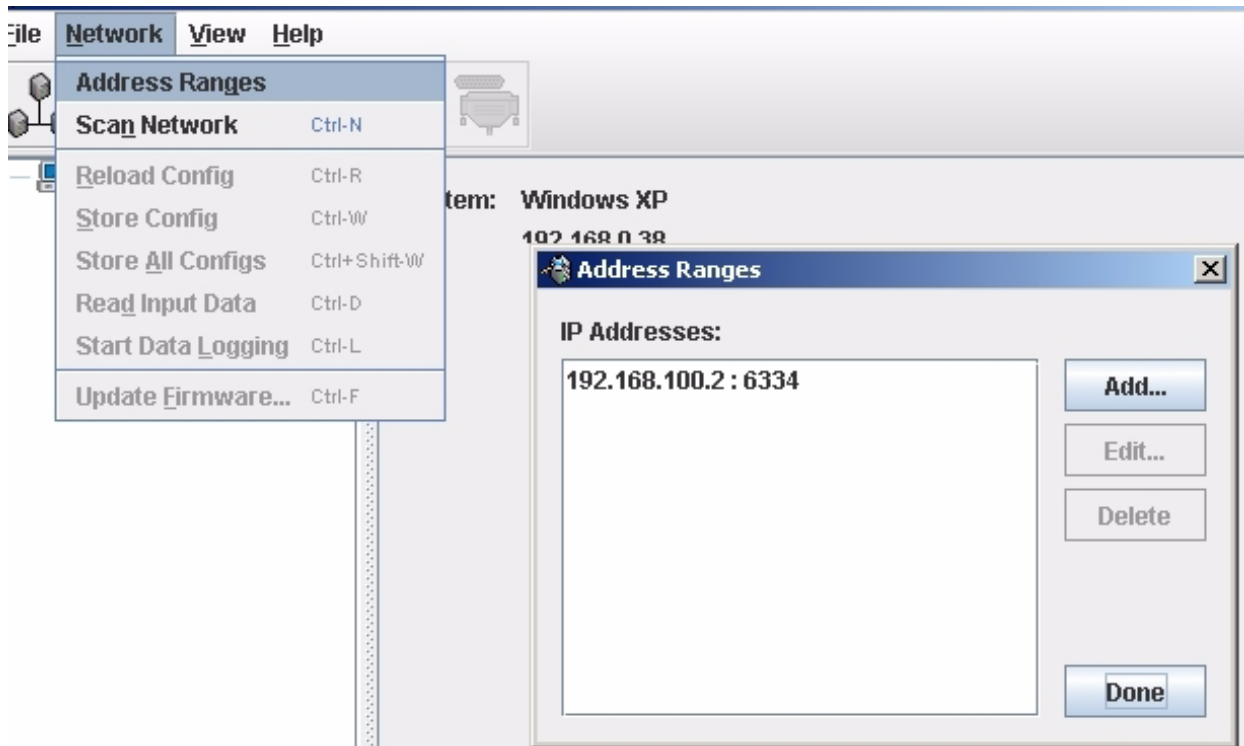


Figure 2-7. Address Ranges to be Scanned

STEP 2: Add the IP address of the PowerDNR system (e.g. 192.168.100.2); click Done.

STEP 3: Now scan the LAN for PowerDNR systems or cubes, using:

Network >> Scan Network

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



STEP 4: Double-click an icon to display its information and list the boards:

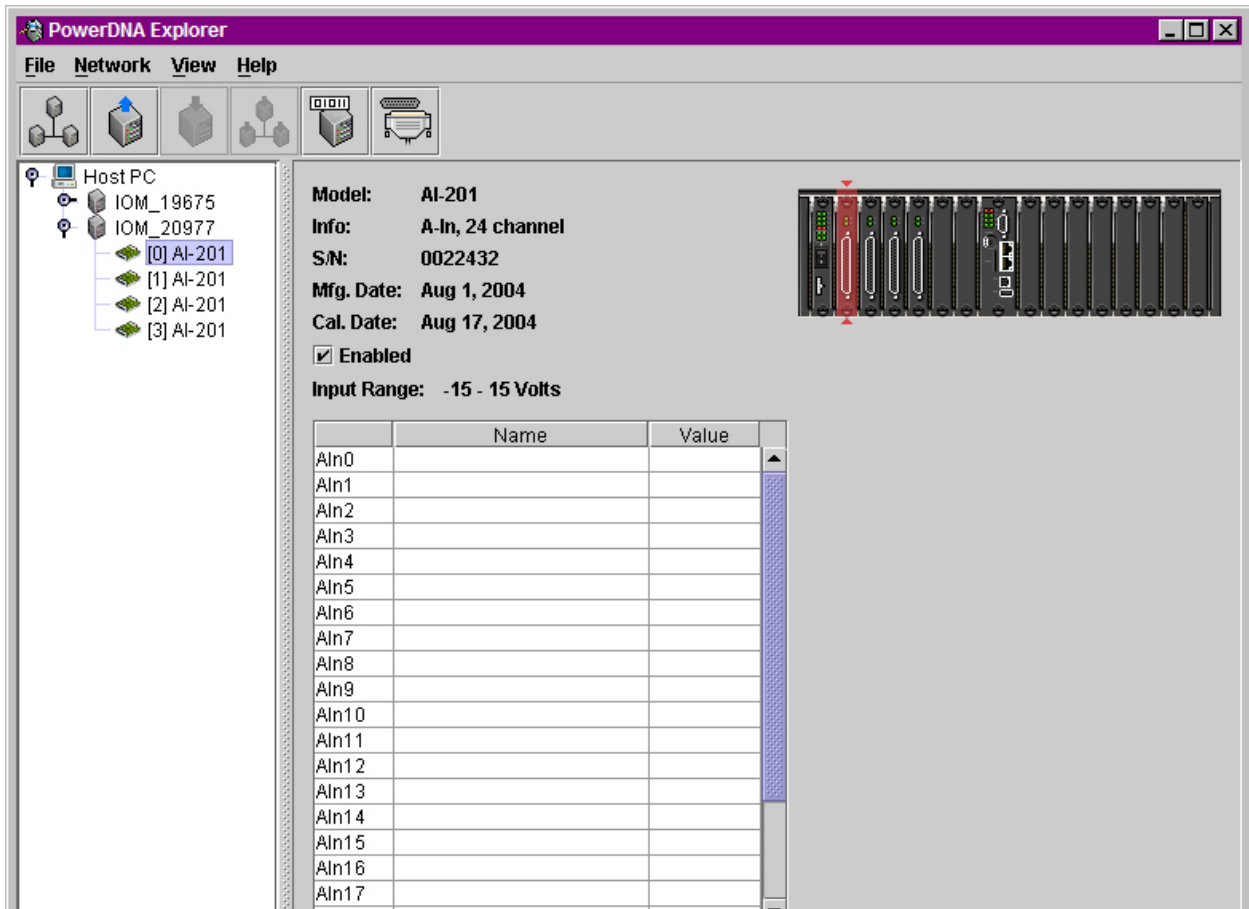


Figure 2-8. Typical Screen for Analog Input Board

The screenshot above is from the PowerDNA Explorer Demo. The “demo” is just a simulator for users without hardware – 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 I/O board, the “Enabled” check box should be set.
- ☒ To read from a board, click the second-to-last button: “Read Input Data”
- ☒ To write to the board, change the value and click the third (or fourth) button with the red arrow on top of the cube: “Store Configuration”. The icon 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 system will then become unreachable.

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



2.4.1 Updating Firmware

Firmware in a DNR-CPU-1000 CPU module 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).



CAUTION!

If you update the firmware in a DNR-CPU-1000, be sure to use the PDNA Explorer from the same release as the 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 system: for the DNR-12-1G, this is the **Firmware_PPC** sub-directory and the rom image file with extension MOT.

NOTE: Before updating the firmware of a system, check the version to determine which update method to use.

- Turn on power to the DNR-POWER-DC module.
- Connect the system to its network.
- Start PowerDNA Explorer on the Microsoft Windows desktop from
Start >> Programs >> UEI >> PowerDNA >> PowerDNA Explorer
- Choose Network ➤ Scan Network
- Select the DNR-12-1G icon you wish to query (by clicking the icon).
- The version is given in the FW Ver field.

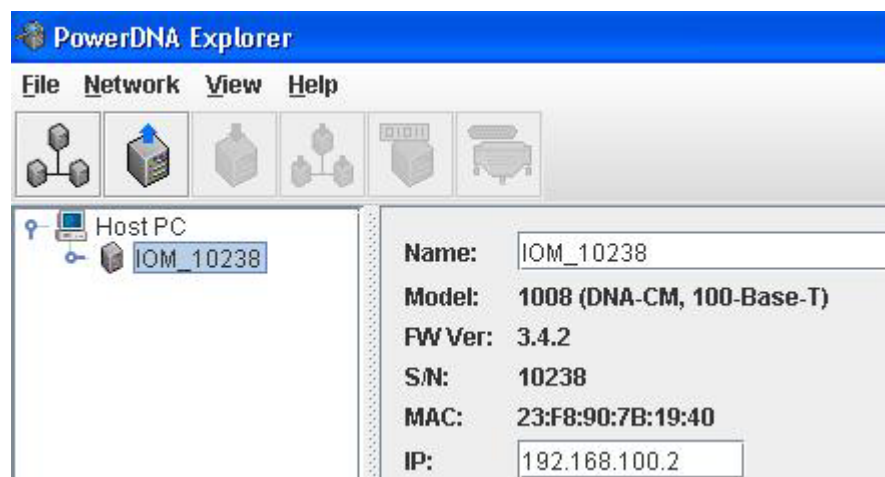


Figure 2-9. Displaying the Version of Your Firmware



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 applicable *Firmware Update Instructions* 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.

2.4.2 Firmware Update Instructions

Before using a new release of the libraries and applications to communicate with your system, you must install the latest version of the firmware onto the DNR-CPU-1000. The version of the firmware *must* correspond to the version of the PowerDNA Software Suite — mismatched versions cause an error.

Instructions for updating the DNR-CPU-1000 via PowerDNA Explorer (over Ethernet LAN line), and over MTTTY (serial line) follow.

Firmware Update over LAN

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

- STEP 1:** Turn on power on the DNR-POWER-DC module.
- STEP 2:** Connect the DNR-CPU-1000 module 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 DNR-12-1G system to be updated.
- STEP 6:** Select *Network >> Update Firmware...* from the menu.

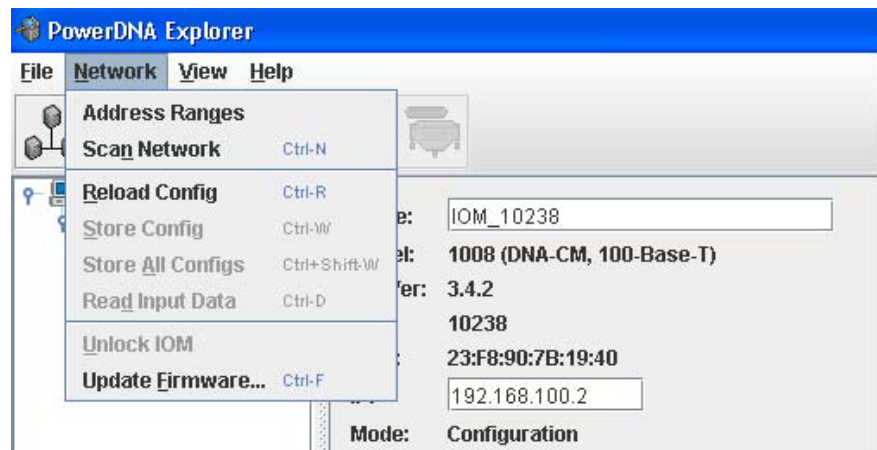


Figure 2-10. Update Firmware Menu Item

- STEP 7:** Click on “Yes” when you see the prompt:
“Are you sure you want to update firmware...”
- STEP 8:** Double-click on the **romimage_X.X.X.mot** (where X.X.X. is the version) file.



- STEP 9:** Enter the password to continue. More information about passwords can be found in the “*Interfacing to the DNR-CPU-1000 Module Using a Serial Interface*” section of this manual. DNR-12-1G systems come with the default password set to `powerdna`.



Figure 2-11. Password Dialog Box

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

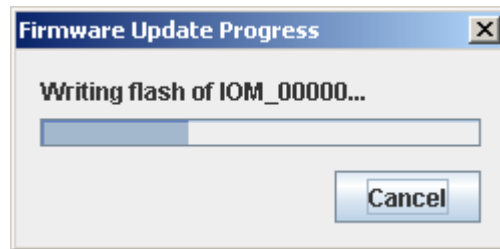


Figure 2-12. Firmware Update Progress Dialog Box

Each system is updated in three steps. First, the firmware is transferred to the system. 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 system 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 system is finished resetting, only the PG light is lit.

Firmware Update via Serial Port

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

- STEP 1:** Establish communication between the PC and a DNR-CPU-1000 CPU over the serial link.
- STEP 2:** Use the hardware Reset switch on the front of the DNR-CPU-1000 Module to reset the CPU Module, or type: `reset all`
- STEP 3:** While the system is starting up again, press ESC to go into `u_boot`.
- STEP 4:** Type the command `erase all` 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\UEI\PowerDNA\Firmware_PPC\romimage_3_x_y.mot*. A progress bar will appear in the lower left corner of MTTTY, indicating progress.
- STEP 6:** Wait for the upload to complete (it may take a few minutes).
- STEP 7:** After the process finishes, enter the `fwjump` command. The DNR-CPU-1000 will then be updated and running the new firmware. At this point, only the PG light on the DNR-CPU-1000 panel remains lit.

2.5 Mounting and Field Connections

You can mount the DNR-12-1G on a flat horizontal surface such as a tabletop or floor, a flat vertical surface such as a wall, or in a standard 19-inch rack. For horizontal surface mounting, use the rubber feet supplied with the standard enclosure or bolt the case directly to the surface. For mounting on a vertical wall surface, attach flanges to both ends of the enclosure with the flanges aligned flush with the rear of the enclosure; then fasten the flanges to the surface with screws or bolts. For mounting in a standard 19-inch rack, attach flanges to both ends of the enclosure with the flanges aligned flush with the front of the enclosure. Then attach the flanges to the rack with bolts.

2.5.1 Physical Dimensions

The DNR-12-ENCL enclosure used in a DNR-12-1G system is compatible with Specification EIA-310-C for 19" Rack Mounting Equipment and is designed to occupy 3U units of vertical space (where 1U is 1.75"). The physical dimensions of the DNR-12 ENCL enclosure are shown below in **Figure 2-13**.

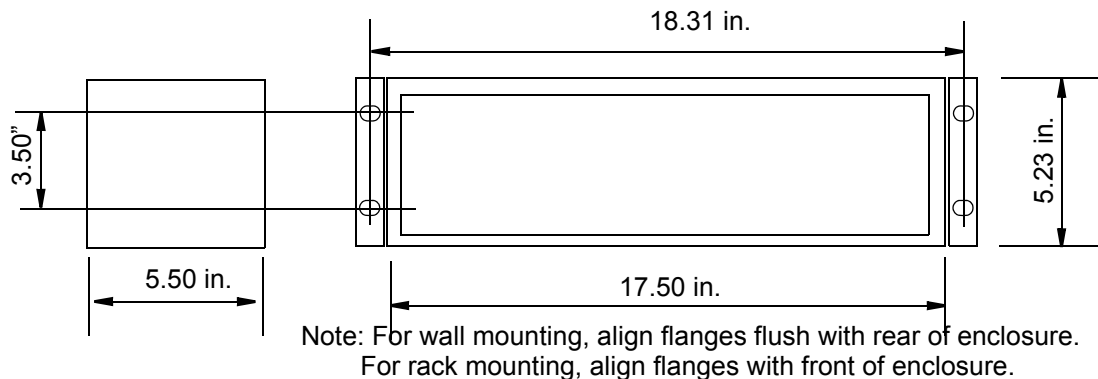


Figure 2-13. Physical Dimensions of DNR-12-ENCL Enclosure



2.6 Wiring

1000Base-T Wiring Configurations

A typical wiring configuration for a 1000Base-T network is shown in the following figure.

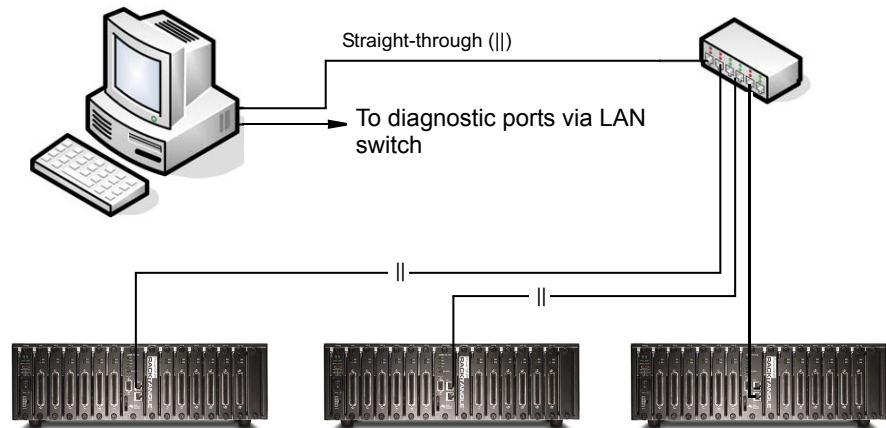


Figure 2-14. System Configuration with LAN Switch



2.7 Peripheral Terminal Panel Wiring

Refer to the applicable I/O board manuals for proper wiring to boards.

2.8 Repairing (and Upgrading) Your DNR-12

DNR-12-1G systems come from the factory fully configured and calibrated. Individual modules are designed for field replacement and are not suited for field repairs. Should you encounter a problem with a DNR-12-1G, you can quickly remove and replace individual boards or other system modules in the field. You can also rearrange the locations of boards within the enclosure at any time. The system automatically senses the slot location of each board. If you want to enhance, repair, or otherwise modify a specific I/O board, however, you must send the module back to the factory or to your local distributor. This process requires that you request an RMA number from UEI before shipping. To do so, you must provide the following information:

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

UEI will process the request and issue an RMA along with an estimate of the work and associated costs required to handle your request.

2.9 Configuring a NIC Port for Diagnostic Mode

The CPU Core Module has two Ethernet ports, NIC1 and NIC2. Either port can be assigned as the Main Operation Port or as a Diagnostics Port.

The main and diagnostics ports are interchangeable. The user application can open both ports independently and use separate handles to access each of them. A port becomes a diagnostics port and prevent changes in the state of the ongoing operation after it is locked -in as a diagnostics port. This allows great flexibility in IOM wiring — if either port or its cabling fails, you can use the other port as the main port.

If all layers are in configuration mode and the lock is not set, the diagnostics port functions as an equivalent of the main port. Any command that can be executed on the main port can be executed on the diagnostics port as well.

The following standard DAQBIOS commands are accessible on the diagnostics port whenever one or more layers are in operating mode:

```
DQCMD_ECHO           // echo
DQCMD_RDCFG          // read configuration (new)
DQCMD_RDSTS          // read status
DQCMD_WRCHNL (selected) //write channel
DQCMD_RDCHNL (selected) // read channel
DQCMD_IOCTL (selected) // ioctl() - low priority command
DQCMD_SETLOCK        // set/release port lock
```

Commands that are capable of changing the state of the running layers will not execute.



To switch a port into diagnostics mode, use the DQCMD_SETLOCK command, as described below:

```
int DAQLIB DqCmdSetLock (int Iom, uint8 Mode, char Password, uint32 *IP
```

Parameters:

```
int Iom          // Pointer to the DQIOME structure
uint8 Mode       // Function mode (lock/unlock/check/diagnostics)
char *Password   // password string; ignored (and can be NULL) if Mode is
                // DQSETLOCK_CHECK
uint32 *IP       // returns the IP address of the locking host if Mode is
                // DQSETLOCK_CHECK
```

<Mode> can be one of the following:

```
#define DQSETLOCK_LOCK      0      // Lock IOM to host
#define DQSETLOCK_UNLOCK    1      // Unlock IOM
#define DQSETLOCK_CHECK     2      // Get locking host IP
#define DQSETLOCK_DIAG     4      // Switch into diagnostics mode
```

To advance a port into diagnostics mode, call this function with the <Mode> parameter set to DQSETLOCK_DIAG. To return a port to normal mode, use the same function call with DQSETLOCK_UNLOCK.

The following table describes the possible states of both ports:

Table 2-1. Port States

Port	LOCK State	First Port (NIC1)	Second Port (NIC2)
First	DQSETLOCK_UNLOCK	Full functionality	Full functionality
	DQSETLOCK_LOCK	Full functionality, locked to the host	All but state change functions
	DQSETLOCK_DIAG	Diagnostic functionality only	Full functionality
Second	DQSETLOCK_UNLOCK	Full functionality	Full functionality
	DQSETLOCK_LOCK	All but state change functions	Full functionality, locked to the host
	DQSETLOCK_DIAG	Full functionality	Diagnostics functionality only

DQCMD_ECHO

This command returns information about the layer(s) installed. Use of this command is described in the API manual.

DQCMD_RDCFG

This command returns the current configuration of the specified layer(s):

```
int DAQLIB DqCmdReadCfg(int Iom, DQRDCFG pDQRdCfg[], uint32 maxsize, uint32*
entries)
```

```
int Iom          // a pointer to the DQIOME structure
DQRDCFGpDQRdCfg[] // structure that contains layer configuration
uint32 maxsize   // number of DQRDCFG structures passed
uint32* entries  // number of DQRDCFG structures returned
```

```
typedef struct (
    uint8DEV;          // device (host fills this field)
```



```
uint8 ss;           // subsystem (host)
uint32 status;      // device status (device returns following fields)
uint32 cfg;         // configuration, including clocks
uint32 rate;        // clock divider in 15.5ns intervals
uint32 clsize;      // size of the channel list
uint32 cl[];        // channel list - variable size
) DQRDCFG, *pDQRDCFG;
```

Note: Use device !=0x80 to indicate that this is the last device in the list.

DQCMD_RDSTS

This command returns the status of the IOM and each and every layer in the stack (upon request):

```
int DAQLIB DqCmdReadStatus (int Iom, uint8 *DeviceNum, uint32 *Entries,
uint32 *Status, uint32 *StatusSize)
```

Parameters:

```
int Iom           // A pointer to the DQIOME structure
uint8 *DeviceNum  // Array of layer numbers to retrieve status from
uint32 *Entries   // Number of entries in DeviceNum array
uint32 *Status    // Buffer to store values received from the device
uint32 *StatusSize // Size of buffer, in 32-bit chunks. Returns number of
                  // 32-bit values copied into Status
```

There are special device numbers to access status of various layers:

0xFE – returns IOM status and status of all layers (note that each layer status is expressed as four 32-bit words. Thus, the maximum size of status packets is $(4 + 14*4)*sizeof(uint32) = 240$ bytes.

0x7F – returns IOM status only (four bytes)

0x0 . . . 0xE – returns status of one of the layers

The status for each layer consists of four 32-bit words, as follows:

```
/* status offsets into devob].status array */
#define STS_STATE      (0) // state of the layer
#define STS_POST       (1) // post status
#define STS_FW         (2) // firmware status
#define STS_LOGIC      (3) // logic status
```

The first word is the state of the layer – what mode of operation it is in, and the lower 8-bits of the timestamp. If the 10us timestamp does not change after each call, the logic is in the inoperative state, as:

```
/* state flags */
#define STS_STATE_TS_SH      8
#define STS_STATE_TS_SH_INS(S,TS) ((S & 0xffff00ff) | ((TS<<8) & 0xff00))
#define STS_STATE_STICKY     (0)
```

The second word describes the status of the layer. It is written when the layer enters initialization mode and remains unchanged until the next reboot. STS_POST_SDCARD_FAILED, STS_POST_DC24 and STS_POST_DCCORE can be changed during operation if the corresponding failure occurs.

```
/* POST status flags */
```



```
#define STS_POST_MEM_FAIL          (1L<<0)      // Memory test failed
#define STS_POST_EEPROM_FAIL      (1L<<1)      // EEPROM read failed
#define STS_POST_LAYER_FAILED     (1L<<2)      // Layer failure
#define STS_POST_FLASH_FAILED     (1L<<3)      // Flash checksum error
#define STS_POST_SDCARD_FAILED    (1L<<4)      // SD card is not present
#define STS_POST_DC24             (1L<<5)      // DC->24 layer failed
#define STS_POST_DCCORE           (1L<<6)      // Core voltage problem
#define STS_POST_BUSTEST_FAILED   (1L<<7)      // Bus test failed (hwtest.c)
#define STS_POST_BUSFAIL_DATA     (1L<<8)      // Bus test failed on data tst
#define STS_POST_BUSFAIL_ADDR     (1L<<9)      // Bus test failed on addr tst
#define STS_POST_OVERHEAT         (1L<<10)     // Overheat detected

#define STS_POST_STICKY
(STS_POST_MEM_FAIL|STS_POST_BUSTEST_FAILED|STS_POST_BUSFAIL_DATA|STS_POST_BU
SFAIL_ADDR
```

The third word contains the logic status flags. They are read and assembled from the various registers of the common layer interface (CLI) upon request. Not all layers implement full functionality and layers operating normally should not show any flags set.

```
/*logic status flags */
#define STS_LOGIC_DC_OOR          (1UL<<0)      // DC/DC out of range (IOM
//also)
#define STS_LOGIC_DC_FAILED      (1UL<<1)      // DC/DC failed (IOM also)
#define STS_LOGIC_TRIG_START     (1UL<<2)      // Trigger event started (IOM
// also)
#define STS_LOGIC_TRIG_STOP      (1UL<<3)      // Trigger event stopped (IOM
// also)
#define STS_LOGIC_CL0_NOT-RUNNING (1UL<<4)      // Output channel list not
// running
#define STS_LOGIC_CLI_NOT_RUNNING (1UL<<5)      // Input channel list not
// running
#define STS_LOGIC_CVCLK_CL0_ERR  (1UL<<6)      // CV clock error for CL0
#define STS_LOGIC_CVCLK_CLI_ERR  (1UL<<7)      // CV clock error for CLI
#define STS_LOGIC_CLCLK_CL0_ERR  (1UL<<8)      // CL clock error for CL0
#define STS_LOGIC_CVCLK_CLI_ERR  (1UL<<9)      // CL clock error for CLI

#define STS_LOGIC_NO_REPORTING    (1UL<<31)     // Installed logic does not
// support error reporting

#define STS_LOGIC_STICKY          (STS_LOGIC_NO_REPORTING)
```

The fourth word contains the status of the firmware. A layer operating normally does not have any flags set except STS_FW_CONFIG_DONE, which means the layer was properly configured before entering operating mode (it is cleared upon re-entering configuration mode) and STS_FW_OPER_MODE, which means that the layer switched into operating mode without any errors.

```
/* fw status flags */
#define STS_FW_CLK_OOR           (1UL<<0)      // Clock out of range (IOM
// also)
#define STS_FW_SYNC_ERR          (1UL<<1)      // Synchronization interface
// error (IOM also)
#define STS_FW_CHNL_ERR          (1UL<<2)      // Channel list is incorrect
```



```
#define STS_FW_BUF_SCANS_PER_INT (1UL<<3) // Buf setting error: scans/  
// packet  
#define STS_FW_BUF_SAMPS_PER_PKT (1UL<<4) // Buf setting error: samples/  
// packet  
#define STS_FW_BUF_RING_SZ (1UL<<5) // Buf setting error: FW  
// buffer ring size  
#define STS_FW_BUF_PREBUF_SZ (1UL<<6) // Buf setting error: Pre-  
// buffering size  
#define STS_FW_BAD_CONFIG (1UL<<7) // Layer cannot operate in  
// current config  
#define STS_FW_BUF_OVER (1UL<<8) // Firmware buffer overrun  
#define STS_FW_BUF_UNDER (1UL<<9) // Firmware buffer underrun  
#define STS_FW_LYR_FIFO_OVER (1UL<<10) // Layer FIFO overrun  
#define STS_FW_LYR_FIFO_UNDER (1UL<<11) // Layer FIFO underrun  
#define STS_FW_EEPROM_FAIL (1UL<<12) // Layer EEPROM failed  
#define STS_FW_GENERAL_FAIL (1UL<<13) // Layer general failure  
#define STS_FW_ISO_TIMEOUT (1UL<<14) // Isolated part reply timeout  
#define STS_FW_CLIO_FAIL (1UL<<15) // CLI or CLO counter does not  
// change (or single-channel  
// ops)  
#define STS_FW_OUT_FAIL (1UL<<16) // Output CB tripped or over-  
// current  
#define STS_FW_IO_FAIL (1UL<<17) // Messaging I/O failed (5xx  
// layers)  
#define STS_FW_NO_MEMORY (1UL<<18) //Error with memory allocation  
#define STS_FW_BAD_OPER (1UL<<19) // Operation was not performed  
// properly  
#define STS_FW_LAYER_ERR (1UL<<20) // Layer entered operation  
// successfully  
  
#define STS_FW_CONFIG_DONE (1UL<<30) //Configuration is completed  
// (no error)  
#define STS_FW_OPER_MODE (1UL<<31) // Layer entered operation  
// mode successfully  
  
/* status helper macros/defines */  
#define STS_FW_STICKY (STS_FW_EEPROM_FAIL|STS_FW_GEBERAL_FAIL)
```

Status bits are divided into “conditional” and “sticky”. Conditional bits are set when a condition arises; they are cleared when the error condition expires. Sticky bits are persistent once set and are cleared by reading their status.

DQCMD_IOCTL

This command is used to retrieve data from the layer. When a port is in diagnostic mode, it returns current data but cannot reprogram the channel list. The channel list is used to inform the handler the ID of the channel from which data should be retrieved.



The following functions, which rely on the DQCMD_IOCTL command for transport, are supported:

Table 2-2. List of Functions and Associated Layers

Function	Associated Layer Type(s)
DqAdv201Read	AI-201 and AI-202
DqAdv205Read	AI-205
DqAdv207Read	AI-207
DqAdv225Read	AI-225
DqAdv3xxWrite	AI-302/308 and AI-332
DqAdv40xRead	DIO-401/405/404/406
DqAdv403Read	DIO-403
DqAdv416GetAll	DIO-416 -- Voltage, current, and circuit breaker state monitoring
DqAdv432GetAll	DIO-432 -- Voltage, current, and circuit breaker state monitoring
DqAdv448Read	DIO-448
DqAdv448ReadAdc	DIO-448 -- Voltage monitoring
DqAdv501GetStatistics	SL-501 and SL-508 -- Received/error counters
DqAdv566GetStatistics	ARINC-429-566 -- Received/error counters
DqAdv601Read	CT-601 -- Counters, states of input lines
DqAdv604Read	QUAD-604 -- Positions, states of input lines

Sequence of Operation

To use the diagnostic port without affecting performance of the main port, UEI recommends that you use the following sequence of operations:

1. Open main port.
2. Open diagnostics port.
3. Perform hardware reset (optional) and re-open ports, if needed.
4. Lock diagnostic port into DQSETLOCK_DIAG.
5. When operation is configured on the main port, read the status of the diagnostics port to verify that the configuration was programmed correctly.
6. Once operation on the main port is started, the diagnostics port becomes available for data retrieval.
7. Read status of the diagnostics port to make sure that all layers of interest successfully entered operating mode without error.
8. In the cycle:
 - a. Retrieve the current status once a second.
 - b. Check the flags for error conditions.
 - c. Retrieve additional data if any flags are set.
9. Stop operation and unlock diagnostics port.
10. Resume normal operation with main port.



Chapter 3 PowerDNA Explorer

The PowerDNA Explorer simplifies configuration and setup of a PowerDNR DNR-12-1G RACKtangle system under Microsoft Windows.

This section describes the various menus/screens in PowerDNA Explorer.

NOTE: The PowerDNA Explorer DEMO lets you safely explore the menus and I/O board screens without using any real hardware.

3.1 The Main Window

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

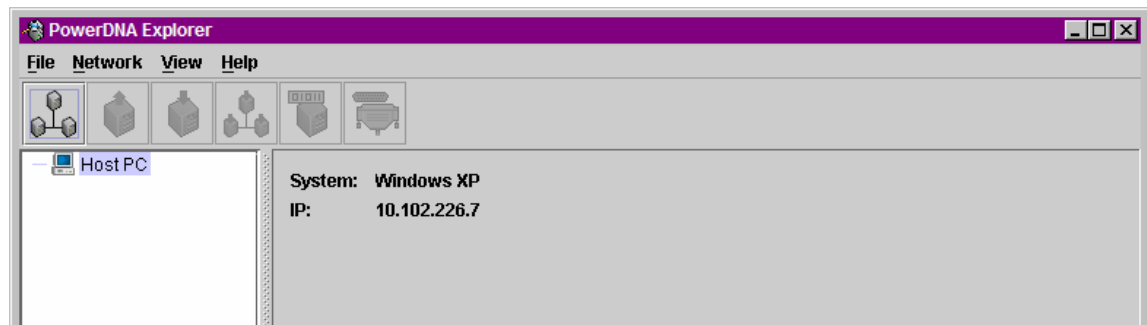


Figure 3-1. PowerDNA Explorer Main Window

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 CPU/NIC Core Module before giving up with an error. It defaults to 100 milliseconds.

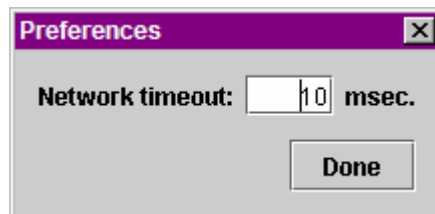


Figure 3-2. Preferences

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.

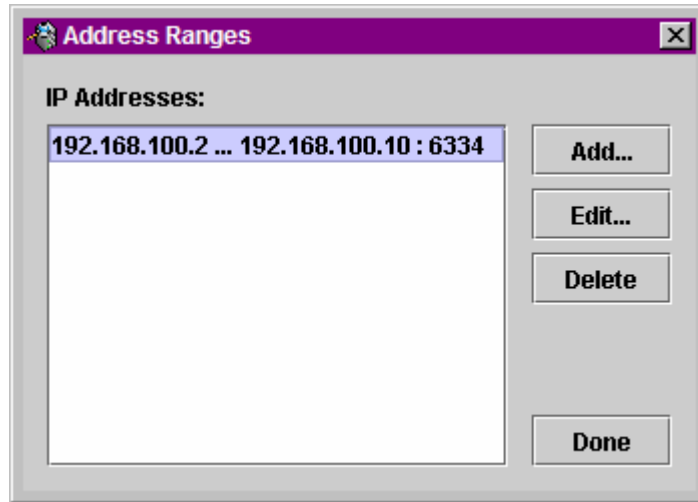


Figure 3-3. Address Ranges Dialog Box

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 to which you can add or delete. This list defaults to a single range item that specifies the range 192.168.100.2 ... 192.168.100.10.

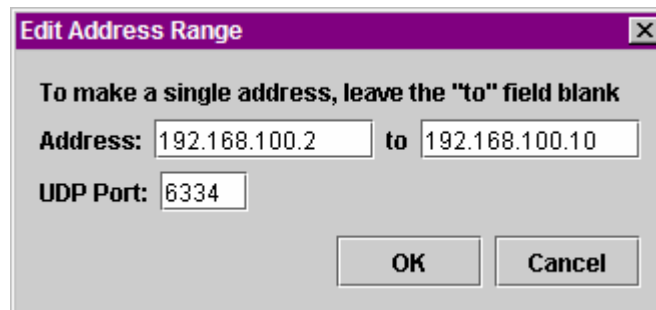


Figure 3-4. Edit Address Ranges Dialog Box

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.

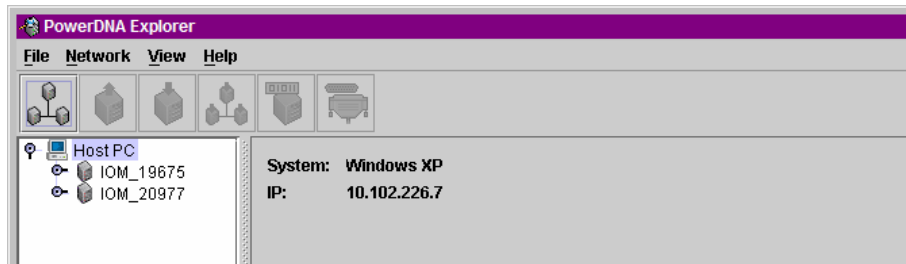


Figure 3-5. After a Network >>Scan Network



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 current settings for the device, after prompting for confirmation.

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

Store All Configs writes all of the changed device 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 board. 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 DNR-12-1G systems if Host PC is selected. It updates only one DNR-12-1G system when a specific unit is specified. More details about this can be found in the section *Updating Firmware in a Version 2.0 PowerDNA system*.

Note that writing certain configuration changes to a PowerDNR system 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. DNR-12-1G systems 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 . . .”



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

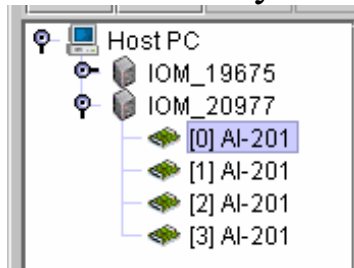


Figure 3-9. Example of the Device Tree

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.

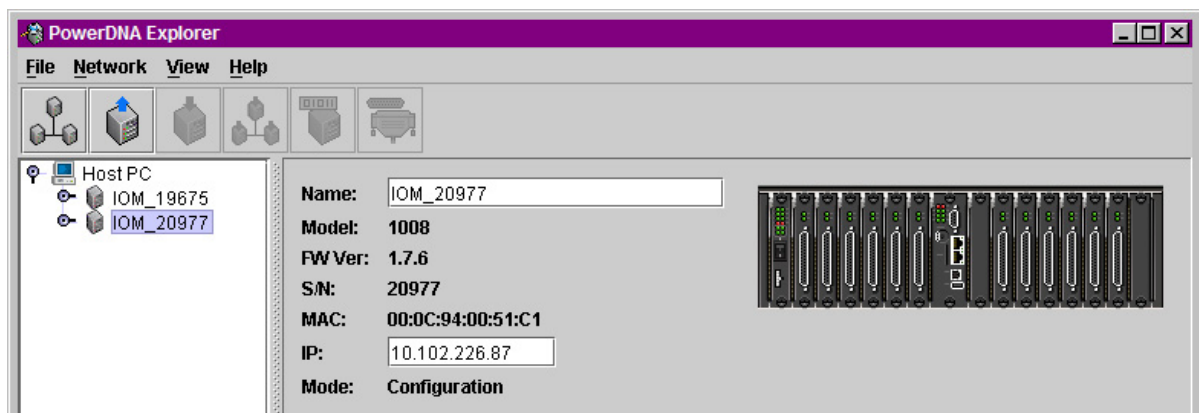


Figure 3-10. Example of IOM Settings Panel for a DNR-12-1G

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 IOM is in: *Initialization*, *Configuration*, *Operation*, or *Shutdown*. These modes are described in the section, *IOM Modes*.

3.2.7.2 I/O Device / Layer Settings

Figure 3-11 shows the screen for displaying I/O device settings.

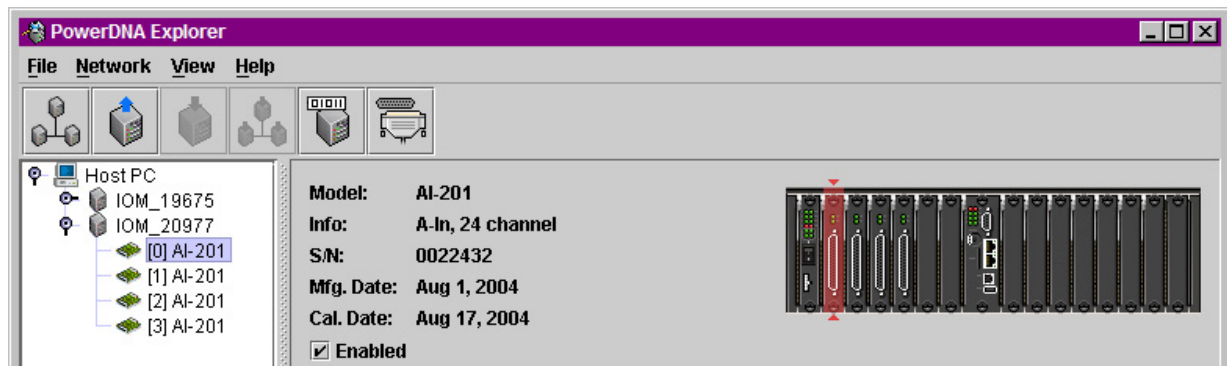


Figure 3-11. Example of I/O Device Settings

Each I/O device has the following settings.

- **Model** shows the model number of the device.
- **Info** shows some key features of the device: A for analog, D for digital, In for input, Out for output, and a number of channels available.
- **S/N** shows the device serial number.
- **Mfg. Date** shows the manufacturing date.
- **Cal. Date** shows the date of the last calibration done.
- **Enabled** is a checkbox which, 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 module, as shown below:

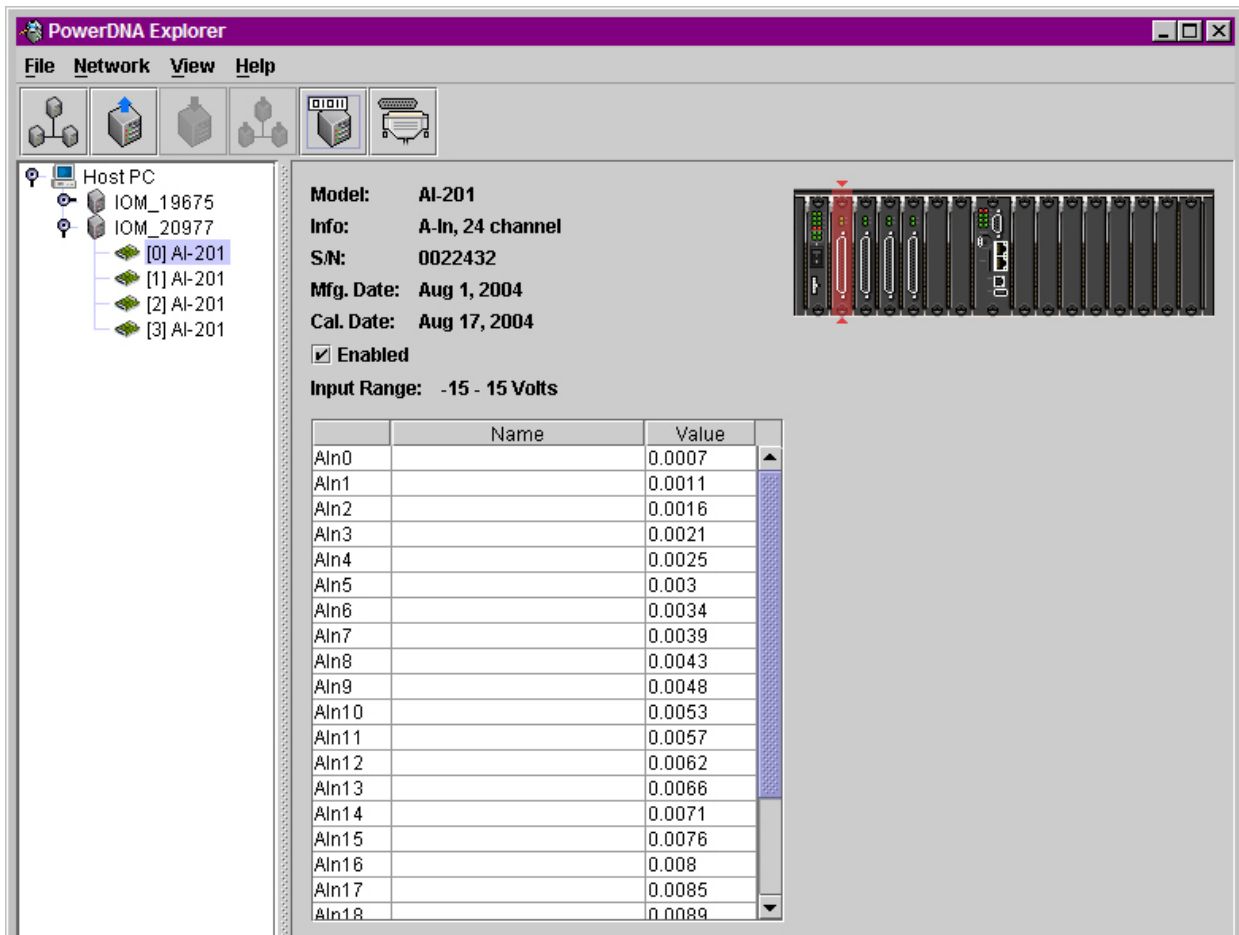


Figure 3-12. Screen from Network >> Read Input Data

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

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 module, using *Network >> Reload Config*.

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

In addition, digital and analog output modules have settings specific to their module types.

3.2.8 Digital Input/Output Module 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 module.



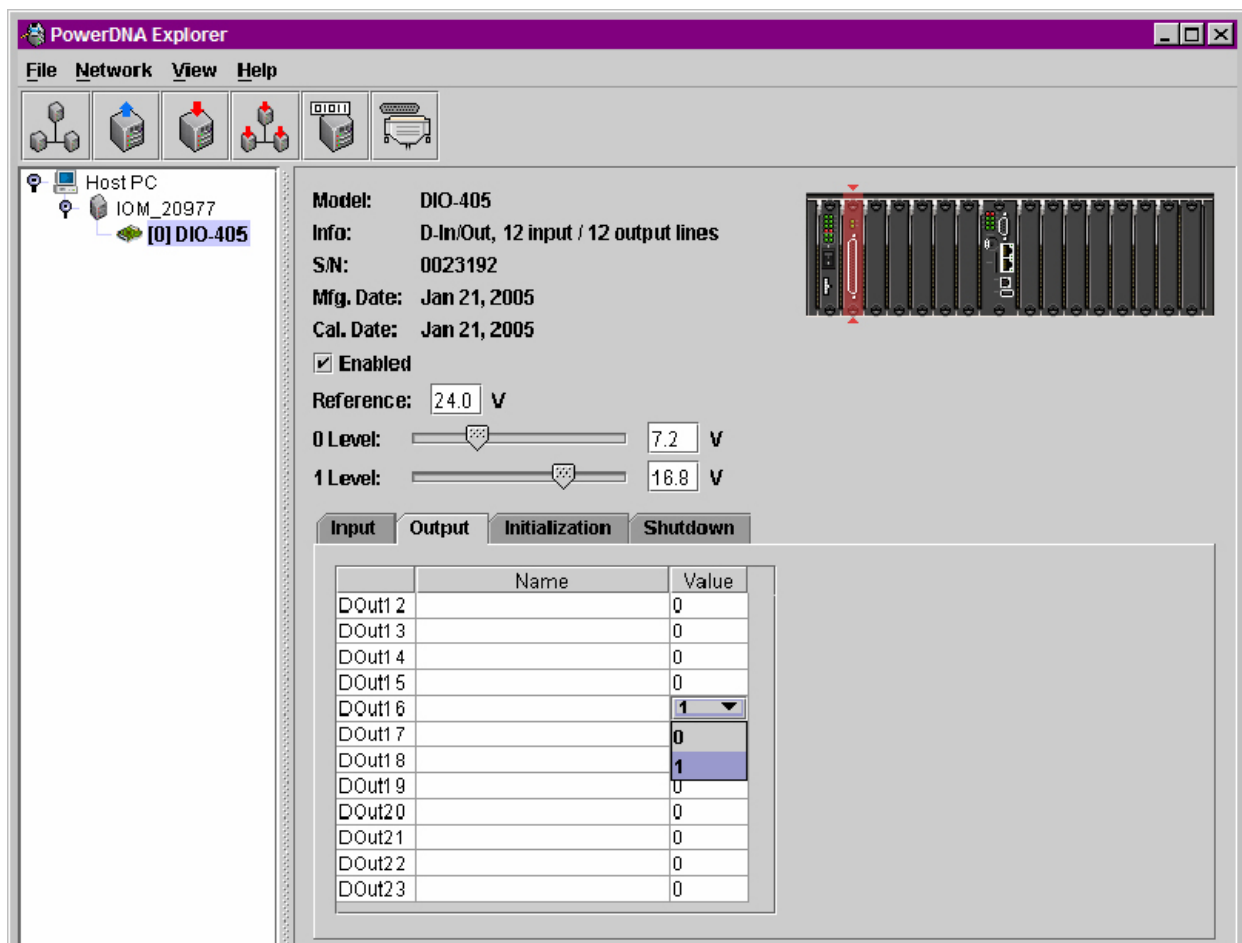


Figure 3-13. Example DIO-405 Inputs



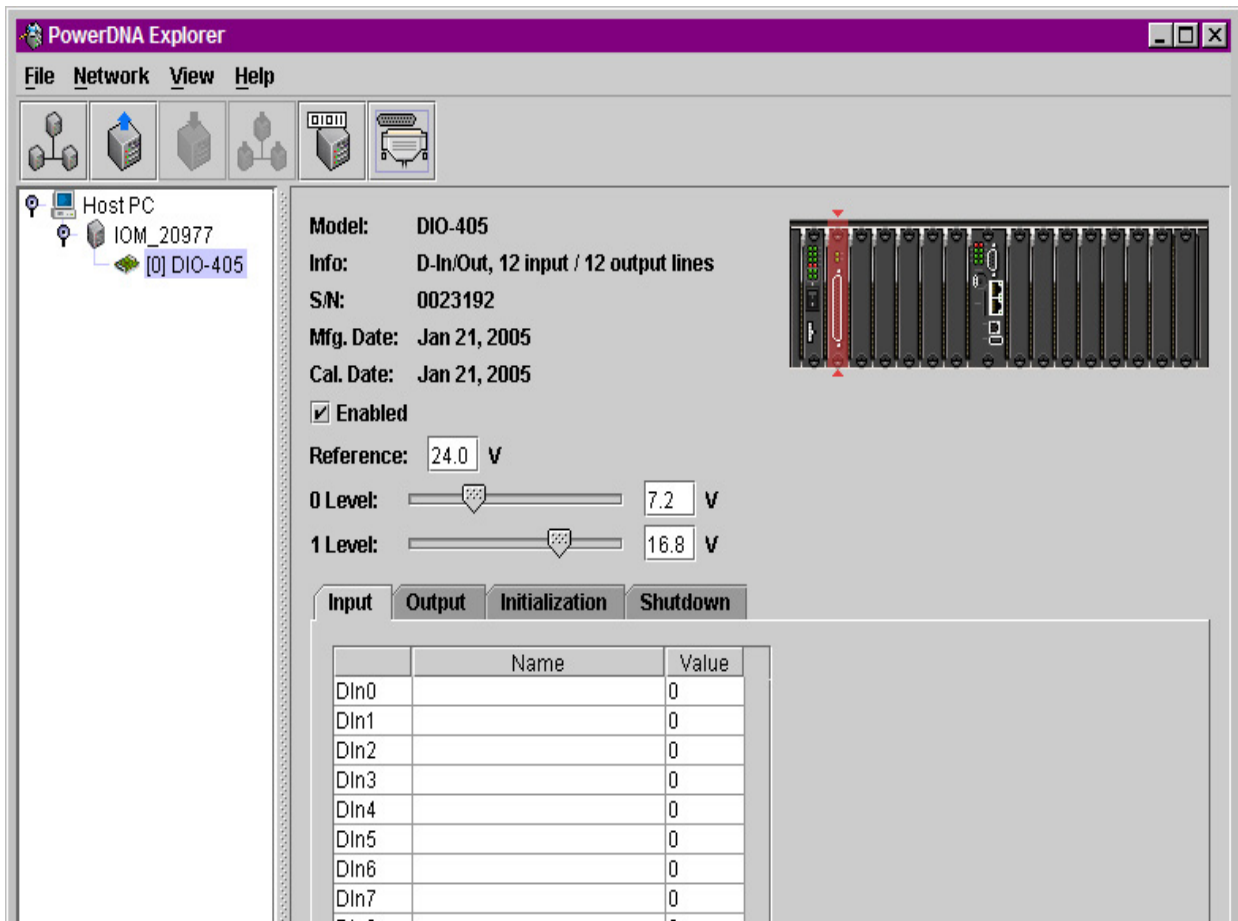


Figure 3-14. Example DIO-405 Outputs

Reference is a reference voltage.

0 level/1 level are hysteresis values described fully in the DIO-401/2/5 manuals.

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 module just has Reference and 0 and 1 Level controls, and Input tab.

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

The DIO-403 module 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.



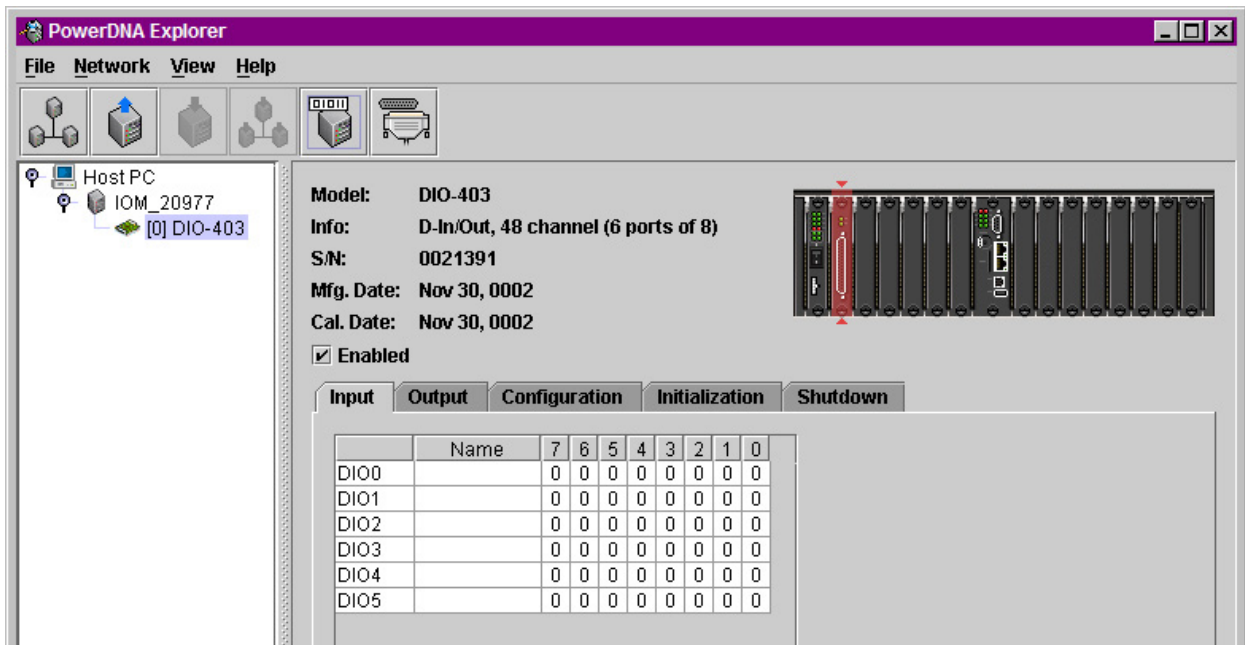


Figure 3-15. Example of DIO-403 Inputs

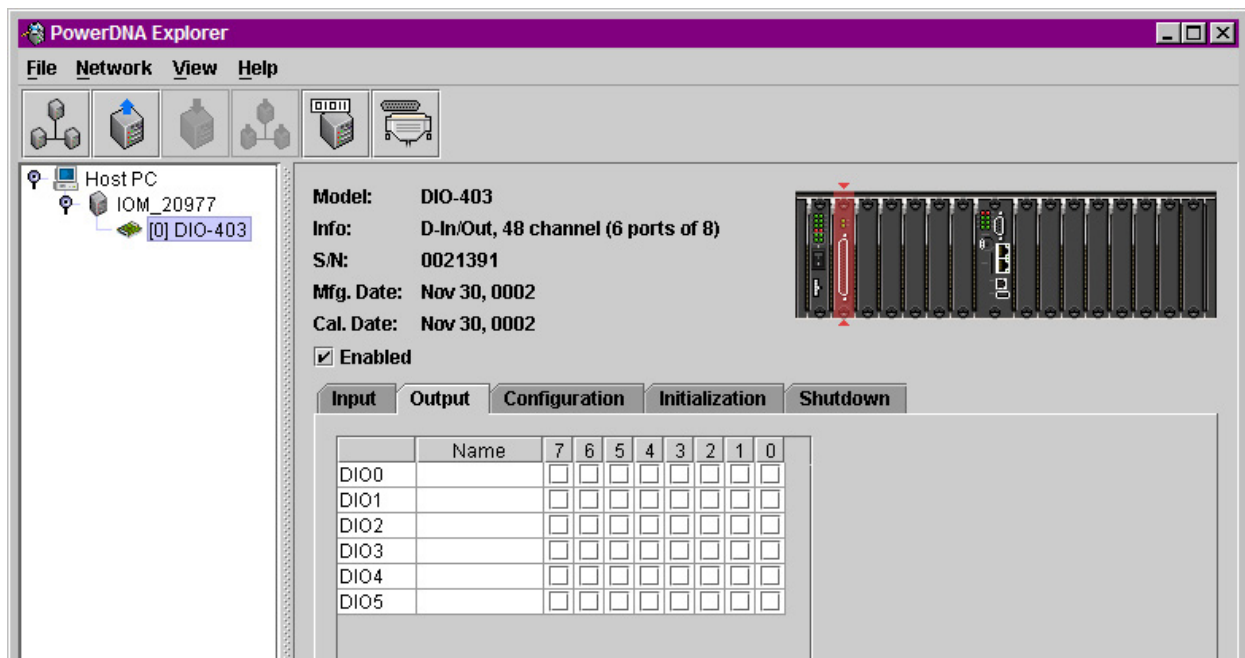


Figure 3-16. Example of DIO-403 Outputs

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).

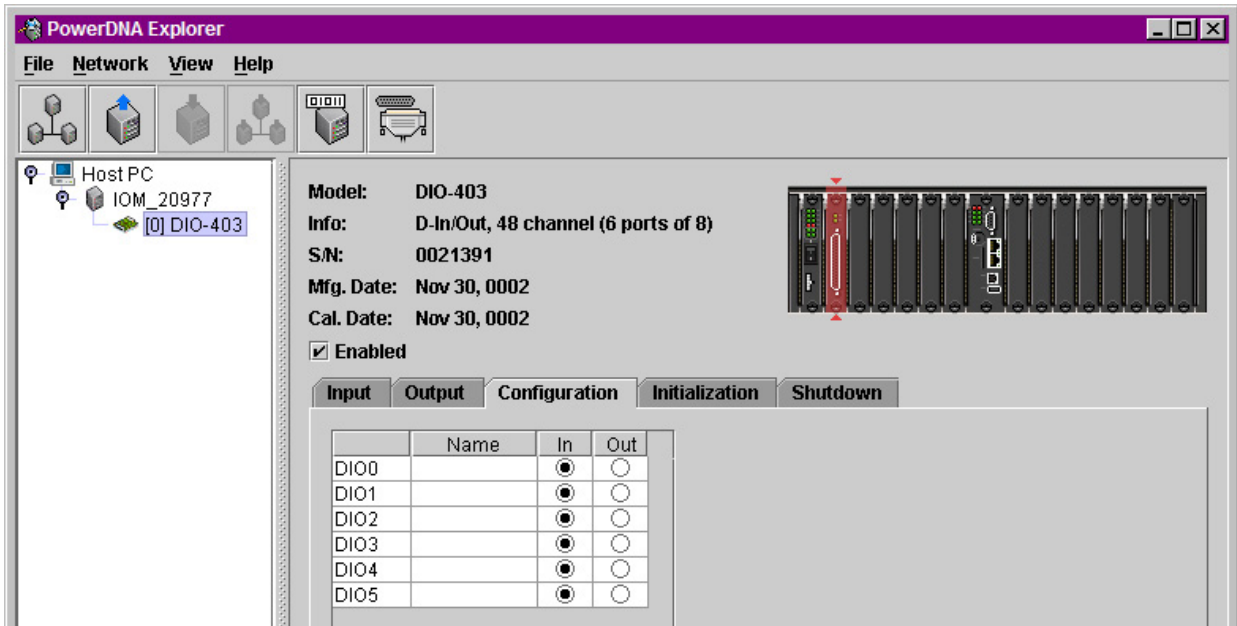


Figure 3-17. Example of DIO-403 Configuration

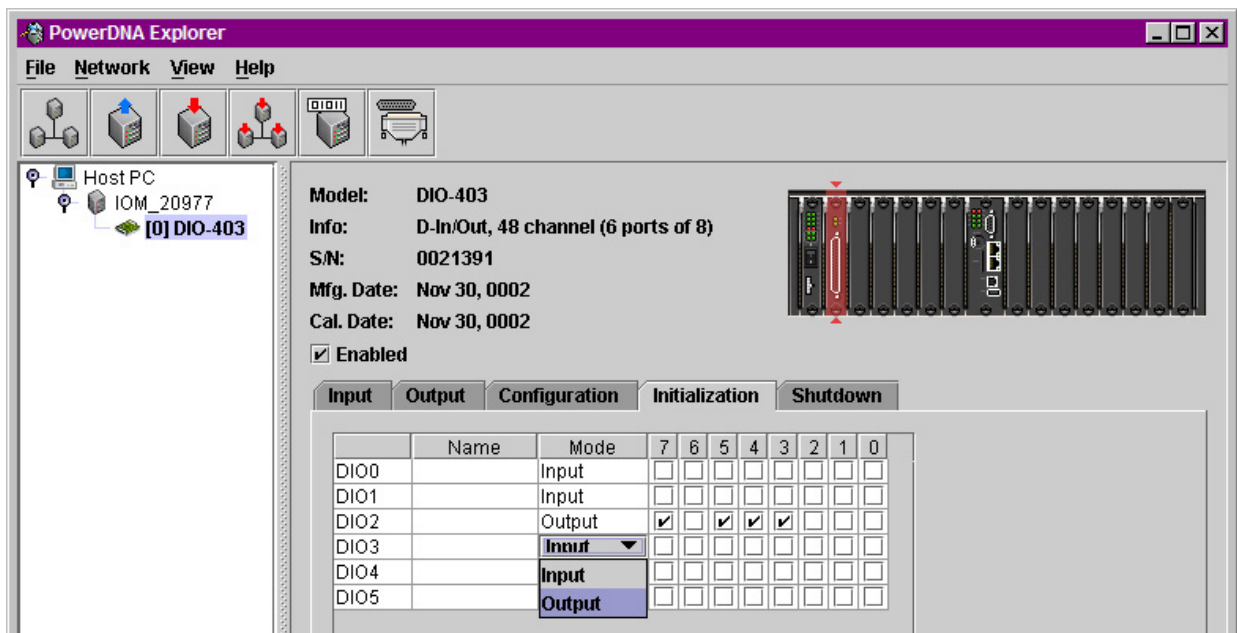


Figure 3-18. Example DIO-403 In/Outputs



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.

Initialization/Shutdown tabs allow you to set 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 Module 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 module.

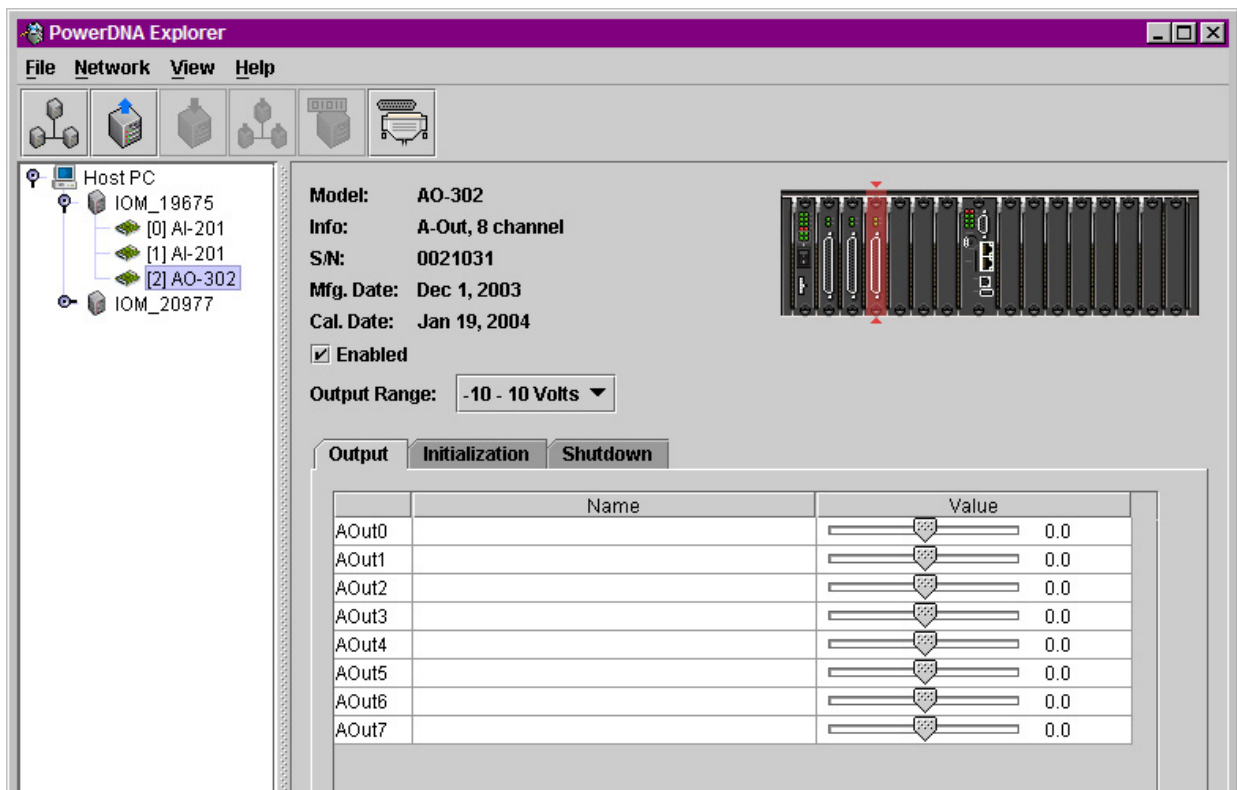


Figure 3-19. Example AO-302 Module



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 module.

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 Module 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 module.

PowerDNA Explorer

File Network View Help

Host PC

- IOM_19675
 - IOM_20977
 - [0] AI-201
 - [1] AI-201
 - [2] AI-201
 - [3] AI-201

Model: AI-201
Info: A-In, 24 channel
S/N: 0022432
Mfg. Date: Aug 1, 2004
Cal. Date: Aug 17, 2004
☒ Enabled
Input Range: -15 - 15 Volts

	Name	Value
AIIn0		0.0007
AIIn1		0.0011
AIIn2		0.0016
AIIn3		0.0021
AIIn4		0.0025
AIIn5		0.003
AIIn6		0.0034
AIIn7		0.0039
AIIn8		0.0043
AIIn9		0.0048
AIIn10		0.0053
AIIn11		0.0057
AIIn12		0.0062
AIIn13		0.0066
AIIn14		0.0071
AIIn15		0.0076
AIIn16		0.008
AIIn17		0.0085

Figure 3-20. Example AI-201 Module



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 Module Settings

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

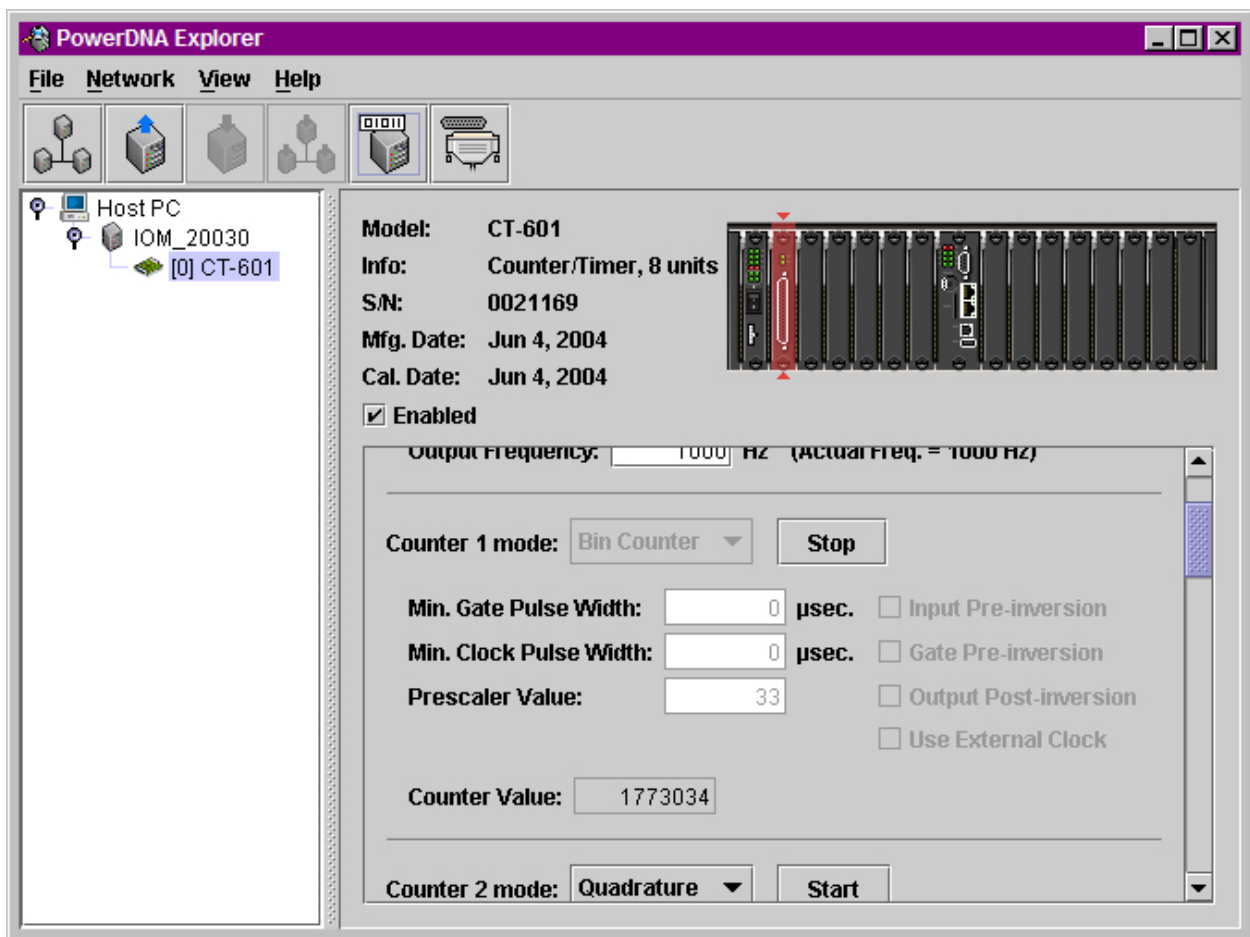


Figure 3-21. Example CT-601 Module

The CT-601 module 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.



Counter 0 mode: **Quadrature**

Min. Gate Pulse Width: μ sec. ☐ Input Pre-inversion

Min. Clock Pulse Width: μ sec. ☐ Gate Pre-inversion

☐ Output Post-inversion

Relative Position: counts

Figure 3-22. Example Quadrature Controls

Counter 0 mode: **Bin Counter**

Min. Gate Pulse Width: μ sec. ☐ Input Pre-inversion

Min. Clock Pulse Width: μ sec. ☐ Gate Pre-inversion

Prescaler Value: ☐ Output Post-inversion

☐ Use External Clock

Counter Value:

Figure 3-23. Example Bin Counter Controls

Counter 0 mode: **PWM**

Duty Cycle: % ☐ Output Post-inversion

Output Frequency: Hz (Actual Freq. = 1000 Hz)

Example Pulse Width Modulation (PWM) controls

Figure 3-24. Example Pulse Width Modulation (PWM) Controls

Counter 0 mode: **Pulse Period**

Min. Gate Pulse Width: μ sec. ☐ Input Pre-inversion

Min. Clock Pulse Width: μ sec. ☐ Gate Pre-inversion

Period Counter: ☐ Output Post-inversion

Positive Count/Period: Frequency: Hz

Negative Count/Period:

Figure 3-25. Example Pulse Period Controls

After setting the configuration for a counter, you can choose *Network* $\rightarrow$ *Store Config* to store the settings on the device. Clicking the Start button will also write your configuration to the module.

Clicking the Start button for a counter will start that counter on the module. 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 module 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.

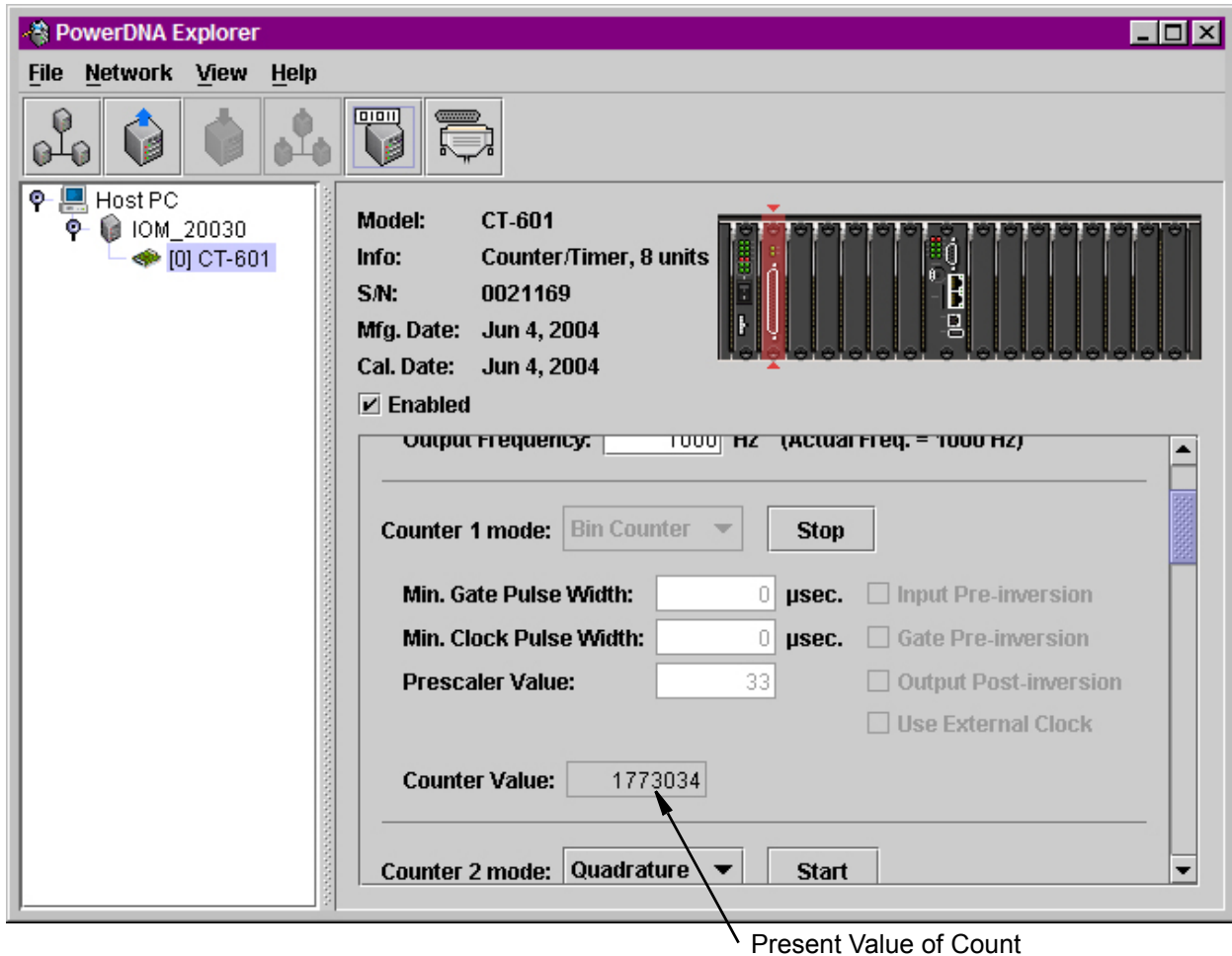


Figure 3-26. Example of Started Counter



Chapter 4 The DNR-CPU-1000 Core Module

This chapter focuses on the device architecture of the Core Module, not I/O modules.

The middle two slots of a 12-slot DNR-12-1G RACKtangle™ Enclosure are occupied by the PowerDNR Core Module, called the DNR-CPU-1000.

The Core Module consists of a Freescale (formerly Motorola) MPC8347 32-bit 400 MHz CPU and peripheral devices (USB 2.0, RS-232, NIC, SD, etc) for use with a Gigabit Ethernet communication network and an internal 66 MHz 32-bit common logic interface bus. The NICs are copper (1000BaseT) interfaces. The module has an RS-232 port used for configuration and also two USB 2.0 ports (controller and slave) for general purpose use (not implemented yet). LEDs on the front panel of each module indicate the current operating status of the device.

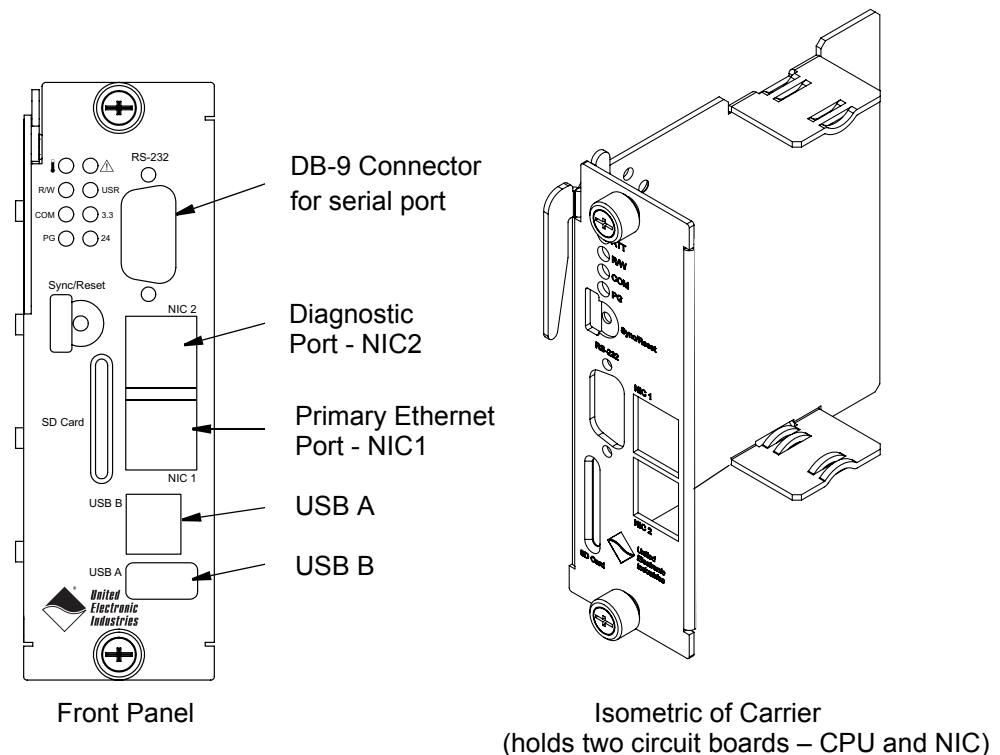


Figure 4-1. PowerDNR Core Module (CPU/NIC) – DNR-CPU-1000



4.1 Device Architecture of DNR Core Module

The DNR-CPU-1000 Core Module architecture can be represented as follows:

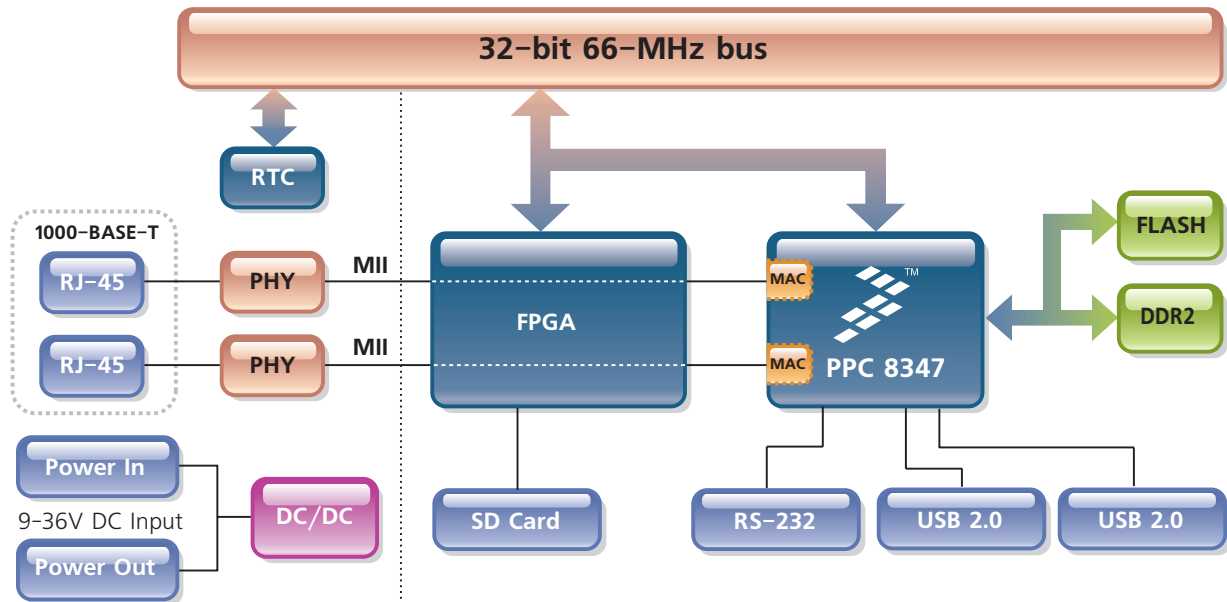


Figure 4-2. FreeScale PowerPC CPU/NIC Controller Architecture

The core of the system is a Freescale (formerly Motorola) PowerPC MPC8347 32-bit 400 MHz processor, which controls the following components:

- Primary Network Interface MII Port – NIC1
- Diagnostic Network Interface Port – NIC2
- RS-232 serial port
- UBS 2.0 dual port (Controller and Slave) (for future use)
- 32MB flash memory
- 128MB of SDRAM (upgradable to 1 Gb)
- SYNC port
- Control logic
- LEDs
- SD Card Slot (Card not included)
- 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 Primary Network Interface MII Port – NIC1 | This port provides communication between the DNR-12 system and the primary LAN network. |
| 4.1.2 Diagnostic Network Interface Port – NIC2 | This port enables the user to monitor the health of the DNR-12 system during operation, using a separate diagnostic port. This port may also be assigned as the primary Ethernet port if NIC1 is not available for use. |
| 4.1.3 RS-232 Port | This port provides a serial communication link between the DNR-12 system and a standard RS-232 terminal. |
| 4.1.4 UBS 2.0 Dual Port (Controller and Slave) | The USB A and B ports are intended for future use and are not software supported at present. |
| 4.1.5 32MB Flash Memory | The DNR-12-1G system is provided with 32MB of flash memory. |
| 4.1.6 128MB of SDRAM (Upgradable to 1 Gb) | The system is supplied with 128MB of SDRAM, which may be upgraded to a maximum of 1GB. |
| 4.1.7 SYNC Port | A high-speed system to system synchronization connector permits triggers or clocks to be shared among multiple systems. Two systems may be connected together directly and larger groups may use the SYNC interface to share timing signals among many racks and systems. |
| 4.1.8 SD Card | A slot for inserting a user-provided Secure Digital card is provided for on-board data storage. It can also store both data and Linux embedded programs using the soon-to-be-released embedded toolkit. Supports FAT12, FAT16, and FAT32 file systems. |
| 4.1.9 LEDs | The operating conditions indicated by the front panel LEDs are described in the figures starting with Figure 1-6 on page 8 and ending with Figure 1-9 on page 10. |
| 4.1.10 Watchdog Timer With Real-time Clock (Battery Backed) | The DNR-12-1G system includes a watchdog timer with battery backed-up real-time clock. |



4.2.11 CPU / NIC Pinouts

Pinout diagrams for the various connectors on the DNR-CPU-1000 and DNR-POWER-DC Modules are shown in **Figure 4-3**.

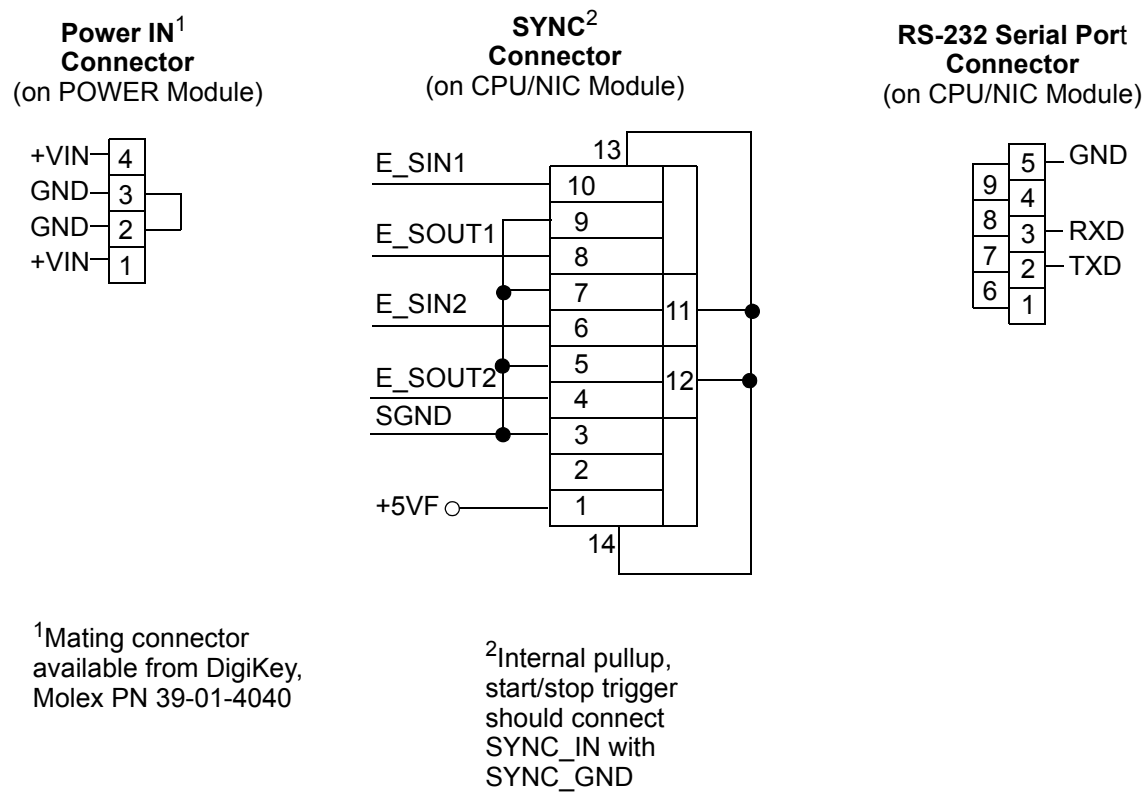


Figure 4-3. CPU / NIC Pinout Diagrams

Chapter 5 Programming Module-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.

The DNR-CPU-1000 Core Module has the following memory map:

Table 5-1. 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	0xA00EFFFF	64kB	EXT_SRAM_ADDRESS
Watchdog timer	0xA00E8000			IOM_WDTIMER – within PLD access space
Processor RAMBAR	0x80000000			
Module – CS2	0xA0000000	0xA00FFFFC	1MB	EXT_DEV_ADDRESS2
Module – CS3	0xA0100000	0xAFFFFFFC	256M	EXT_DEV_ADDRESS3
Flash (parameters)	0xFFC00000	0xFFC0FFFF	64kB	Parameters (64 sectors)
Flash (firmware)	0xFFC10000	0xFFEFFFFFFF	3MB	Firmware (3MB – 64kB)
Flash (U-Boot)	0xFFF00000	0xFFFFFFFF	1MB	U-Boot

Two address ranges are interesting for host software:

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

5.2 Startup Sequence

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 the “fwgo” parameter is set to “autorun”, the monitor waits for three seconds for you 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 new firmware, 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 for firmware to be copied and started from the proper address:

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

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

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

“**fwgo**” defines whether the monitor should load firmware or 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. The firmware is compiled to run from this address.

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

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

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

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 the exception table, and starts execution.

5.3 Setting Core Module Parameters

There are two ways to set up Core Module (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, you should connect a 9-wire serial extender cable to the DNR-12 CPU/NIC module (male plug connector) and your PC COM1 serial port (female connector). Some cables have female-to-female connectors, so you may have to 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\\UEI\\PowerDNA\\Firmware\\mttty.exe* Then click *File>>Connect*.



Once a connection to the PowerDNR DNR-12-1G system is established, tap “Enter” once. The DNR-12-1G 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

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

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 DNR-12-1G parameter.

To change parameters, use the “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, you should reset firmware (start firmware execution from the beginning without full hardware reset), as follows:

```

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 (such as `mr`, `mw`, `set`, and `store`) 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 super-user level password. First, you'll be asked about old super-user password and then (if it matches) to enter the new super-user password twice.

DNR-12 systems come with the default password set to "powerdna".

Some DaqBIOS commands require clearing up user or super-user password. 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 modules found and initialized by firmware along with assigned device numbers.

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

Priority determines the order in which 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 module parameters – serial number, option, etc. We do not recommend use of this command in the field.

"simod 1" allows module calibration. Different modules 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 module warranty is void.

5.3.1 Setting Parameters Via Serial Interface

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). A valid value is 0x3012.

“**Serial**” sets the DNR-12 serial number (factory programmed, do not change)

“**MAC**” sets the DNR-12 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.

“**runtype**” for the DNR-12 should be 2.

“**portnum**” and “**umports**” should be 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 DNR-12 system is used over the UDP protocol or from the host.

“**IP**” specifies the IOM IP address. This is the most important parameter the user must change to allow the DNR-12 system to be visible on the network. The DNR-12 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 DNR-12 should send an IP packet if a requested IP packet exists outside of the DNR-12 network (defined by the network mask). Ask your system administrator if you use your DNR-12 on the office network.

“**netmask**” specifies what type of subnet the DNR-12 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 DNR-12 without a previous request from the host side. If the DNR-12 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 you want to connect a DNR-12 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

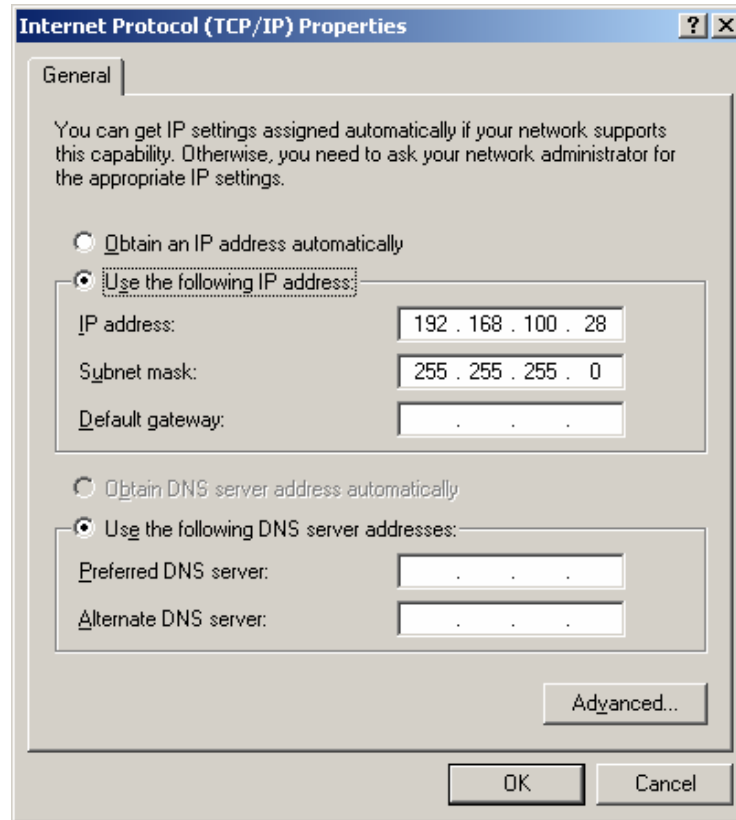


Figure 5-1. TCP/IP Properties

Set DNR-12 system 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 DNR-12 system.

DNR-12 systems 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 DNR-12 enclosure along with factory-set MAC address.

After the IP address is set, you can establish communication with the DNR-12 system using PowerDNA Explorer.

5.3.2 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 Module (Layer) Interface

The Common Layer Interface is the protocol used in a PowerDNR system for communication between the IOM and its layers (I/O boards).

5.4.1 Channel List

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

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

```
// 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 do
// 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, as follows:



```
#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 different 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 modules

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

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

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

DQ_LNCL_NOWAIT – causes the module 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 you need to write data to the output channels sequentially and update them at the same time. In this situation, you should use the **DQ_LNCL_WRITE** flag for every entry. To update all outputs with previously written values, combine the **DQ_LNCL_WRITE** flag 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 module type. See powerdna.h for module-specific gain macros.

5.4.2 Configuration Flags

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

```
// Standard part (lower 16 bits) of module configuration word
// Please notice that for multiple-subsystem modules you 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 module to return raw
                                   // unconverted
                                   // data
#define DQ_LN_TMREN       (1L<<11) // enable module periodic timer
#define DQ_LN_IRQEN       (1L<<10) // enable module 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 module.

DQ_LN_ENABLE enables all operations within the module

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 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 you are interested in a phase shift between channels.

You 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

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 module does not receive output data, it will recycle old data until new data is available; otherwise, the module will stop at the last value output

DQ_LN_GETRAW – tells the module to return uncalibrated unconverted data. This flag makes sense only for modules with software calibration (AI-225, for example). Moving calibration and conversion of data to host unloads the 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 module to store one channel list worth of data in the buffer. On a timer tick, the firmware transfers this data from the module output buffer to the packet. This function is used when the hardware allows only a selected set of update rates, but you need something in between. For example, AI-225 can convert data with fixed frequency equal $6.875\text{Hz} * 2^n$, where $n = [0..9]$. To receive an exactly 500Hz data stream from this module, specify that this module be updated upon a timer tick.

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

5.4.3 EEPROM User Area Access

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

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

E²PROM is divided into certain access areas (some of them can be missing in different module 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 module 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 all 2048 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 module calibration information. Firmware writes this information automatically upon entering initialization mode.

OPMODEPRM_xxx_ contains module 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 - module 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 module 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 module type. Only 512 bytes are allocated for channel names. Thus, AI-205 module (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.

5.4.4 PowerDNA Module Signaling

This section defines procedures for setting up triggering, synchronization, and clocking lines.

Most PowerDNA modules have the following interconnection diagram:

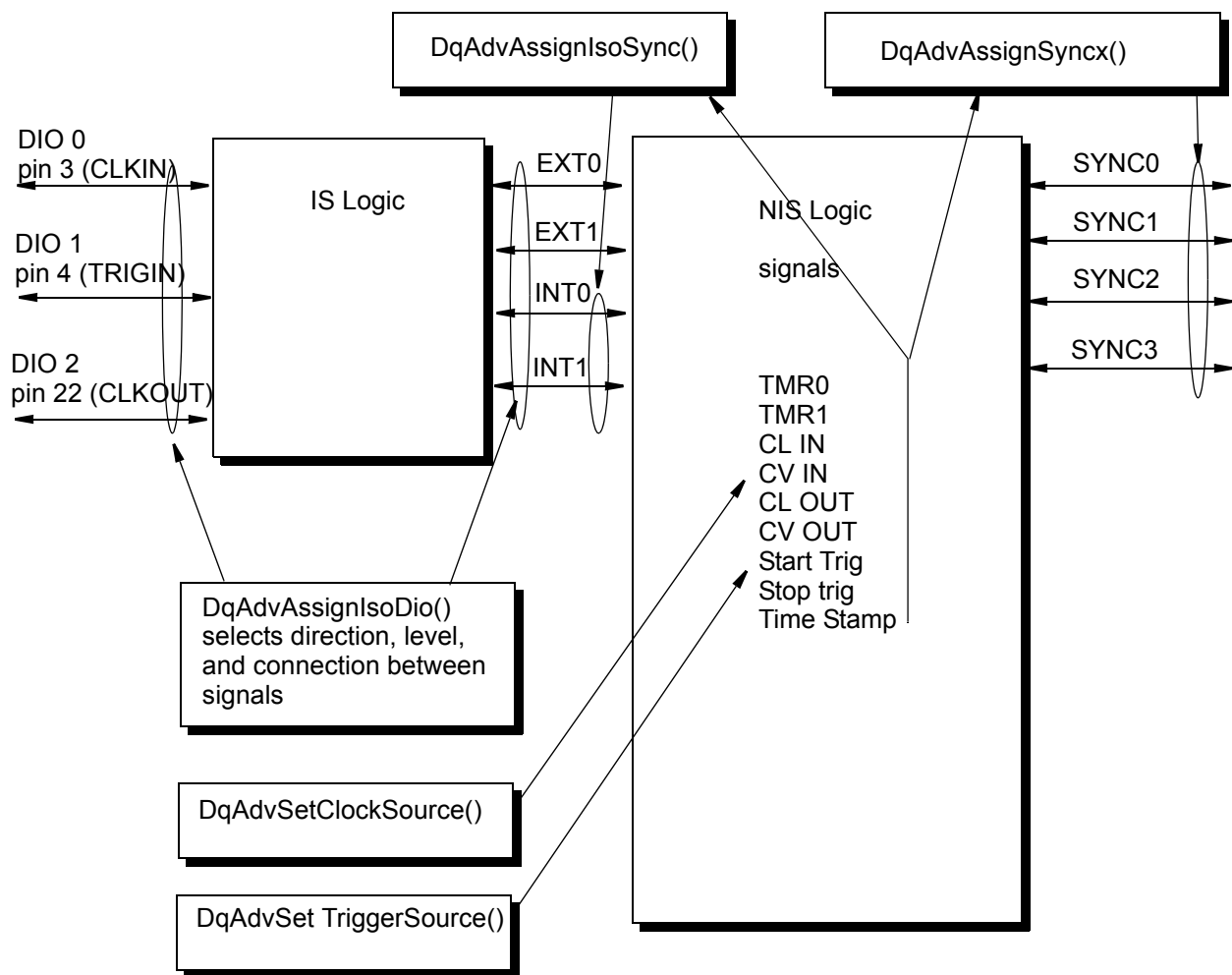


Figure 5-2. Core Module Interconnection Diagram

- 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. Clock can be selected from internal sources, EXT_x lines (signals from the isolated side) and SYNC_x interface signals (inputs)

- `DqAdvSetTriggerSource()`

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

- `DqAdvAssignIsoDio()`

This function selects direction and signal assignment for external DIO line. EXT_{0/1} lines are assigned to DIO_{0/1} 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 module logic will be assigned to INT_x lines. Signals can be selected from internal clock sources and SYNC_x lines.

- `DqAdvAssignSyncx()`

This function selects a signal for each of the SYNC_x lines. When a SYNC line is selected, it switches to the output state. All other modules “listen” to this command on the system bus and release that SYNC line from use (switch to the input mode). This organization prevents two modules 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: Note that to take advantage of using external clocks for the module 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 module clocking.

The same interface applies to the CPU module. The CPU module 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.

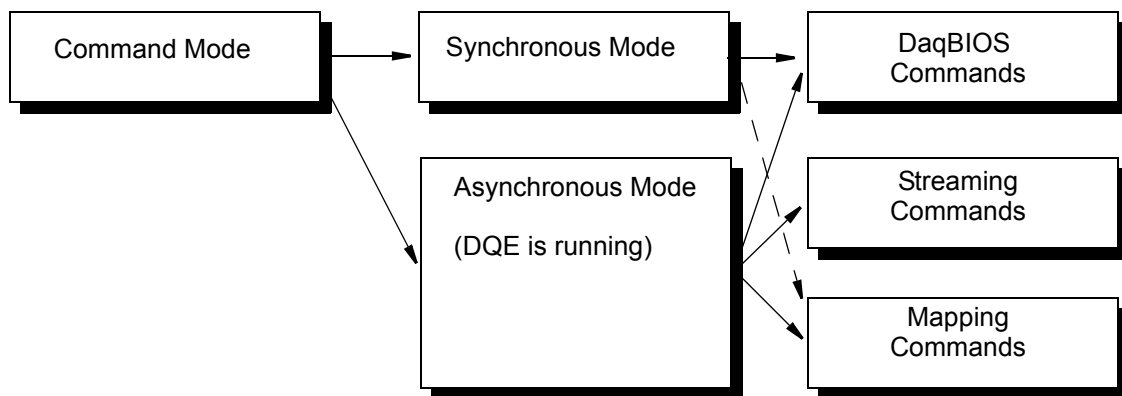


Chapter 6 Host / IOM Communication

6.1 Host / IOM Communication Modes

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

- DaqBIOS Command API (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)



NOTE: DQEngine, used only on desktop operating systems such as Windows and Linux, takes care of sending packets at a periodic rate, receiving incoming packets in the background, and automatically correcting transmission errors. It is needed on desktop OSs to ensure that DMap and ACB buffers are periodically refreshed, but is not required on realtime systems. The DQE needs its own thread and the user must synchronize it with its own processing loop.

On RTOSs such as QNX, RTX, and VxWorks, it is the responsibility of the user to send and receive packets at a periodic rate. This is much easier to achieve on an RTOS than on a desktop OS.

Figure 6-1. Communicating with an IOM

6.1.1 Additional Modes

In addition to the basic modes listed above, subsets of these modes are also available for use, as follows:

- Messaging — asynchronous, buffered, messaging data format
- Mapped Messaging (M3) — asynchronous, mapped I/O, messaging data format

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.



Some important characteristics of these additional modes are:

- In ACB mode, you transfer data 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, you can transfer data to/from multiple Dmap or Vmap I/O boards 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.
- In Mapped Messaging (M3) mode, you can transfer multiple data values per channel to/from multiple M3 boards all in one packet and also run multiple M3 boards at different speeds.

6.1.2 Synchronous and Asynchronous Modes

In synchronous mode (point-by-point), the host sends a request and waits for a reply, and then sends another command.

In asynchronous mode, the host sends requests on ticks of the timebase timer. Asynchronous mode takes care of re-requests in the case of packet loss or network collision. In asynchronous mode, you can work the same way as in synchronous by sending request after request and processing packets yourself. However, we encourage you to use asynchronous mode for streaming and to design your application around this paradigm.

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

For real-time response under the control of a real-time OS, you can perform mapping using synchronous mode commands or use the FIFO interface to retrieve/send the stream of data. As synchronous mode does not have error correction and data-flow control built-in, you must perform these tasks yourself.

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 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.

6.1.3 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 asynchronous buffered modes are called Advanced Circular Buffer (ACB) and Burst Mode.

6.1.4 Burst Mode

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

6.2 Advanced Circular Buffer (ACB)

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.



An Advanced Circular Buffer (ACB) is a regular FIFO represented as a circular buffer. What you do with the data once it arrives in host memory can also have a major impact on system performance. The PowerDNA software uses the concept of an Advanced Circular Buffer. When combined with applications tuned to take advantage of this flexible buffering mechanism, the system as a whole runs much more efficiently.

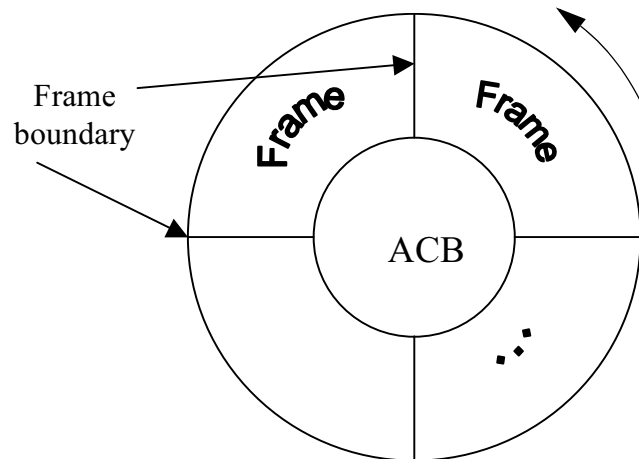


Figure 6-2. ACB and Ring Buffer Overall Structure

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 irrespective of whether or not the application reads the data. In this mode, a buffer overflow never occurs. It's best 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 might 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 a more efficient performance, is that the logical buffers in the queues cannot be dynamically allocated or freed and their order is fixed.

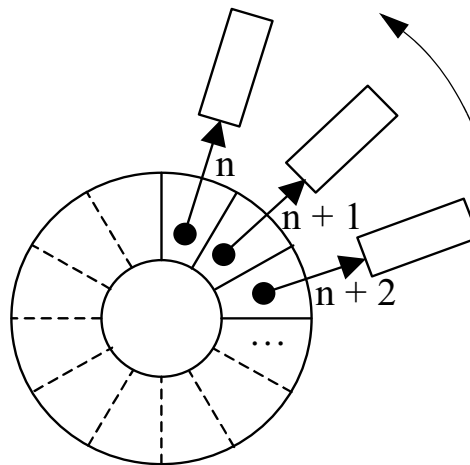


Figure 6-3. Packet Ring Buffer

The Ethernet UDP protocol used to transfer data is connectionless and unreliable. Older packets might come first while 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.

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 we receive a higher-numbered packet than expected. (In **Figure 6-3**, 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 there is a chunk of contiguous data available at the moment of execution.

When the writer thread writes converted data from the ring buffer into the ACB, it checks for frame and buffer boundaries as well as for error conditions. If those conditions exist and the user application has signed up to receive events upon one of these conditions, the writer thread sets up an event (synchronization primitive). Upon receiving an event, the user application can determine what condition caused the event to fire.

An ACB generates the following events:

- **DQ_eDataAvailable** is generated when writer thread transfers any data from the ring buffer to the ACB. In case of output operation, `DQ_eDataAvailable` is set when there is a place in the buffer into which new data can be put.
- **DQ_eFrameDone** is set when incoming data crosses a frame boundary — when the writer thread has contiguous data in the ring buffer and transfers it into the ACB. In the case of output operation, the reader thread takes data from the ACB, converts it, and puts it into the output ring buffer. Thus, at the beginning of the output operation, the `DQ_eFrameDone` event is set quite frequently while data is transferred from the ACB to the empty ring buffer.
- **DQ_eBufferDone** is set in Single mode when the buffer becomes full on input or empty on output. Normally `DQ_eBufferDone` is accompanied by a `DQ_eStopped` flag.
- **DQ_ePacketLost** is set when one or more packets is lost and unrecoverable. In case of input, the DQE tries to request the missing packet for a defined number of times. If this effort fails, the `DQ_ePacketLost` flag is set. The DQE fills the place allocated for the missing packet in the ring buffer with zeroes or with a user-supplied pattern. Then, it releases this packet to the writer thread. Similar processing happens when the IOM replies to the host with the `DQERR_TOOOLD` flag set. `DQERR_TOOOLD` means that the requested packet of data is already overwritten with the new data.

However, if the IOM locates the missing packet, it tries to request this packet from the host for a defined number of times. If the packet is still missing, the IOM sends a message packet to the host with a `DQERR_LOST` flag. This means that the IOM cannot recover the missing packet and should skip this packet in the output ring buffer.



- **DQ_ePacketOOB** (packet is out of bounds) is set when a packet received by the host is so far off that the host cannot insert this packet into the ring buffer. In the similar case for output, when the IOM cannot insert a packet of data received from host into its ring buffer, the IOM replies with a **DQERR_TOOOLD** flag.

6.3 DMap Structure and Function

DMap is one of the operating modes of PowerDNA. It continuously refreshes a set of channel data that can span multiple I/O boards at a specified rate paced by the IOM's hardware clock.

In DMap protocol, the user application can be informed either that the exchange took place (**eDataAvailable** event) or that the exchange failed (**ePacketLost** event).

In cases in which an exchange took place, the data from the output buffer was delivered to the IOM, outputs were updated, and data in the input buffer was updated by the values retrieved from the input channels of the IOM.

In cases in which an exchange failed, the DMap protocol does not allow the system to recognize which part of the update cycle caused the problem. The packet could have been lost in transmission from the host to the IOM or from the IOM to the host. What's important is that a control application relying on DMap should wait for the next cycle with the latest data before making any changes to the output. There is normally no reason to re-retrieve lost samples, however, because control applications usually must keep up with the latest state of the inputs. In other words, timely delivery of the freshest data has the highest priority and the lost data can be easily ignored.



6.3.5 DMap Functional Description

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

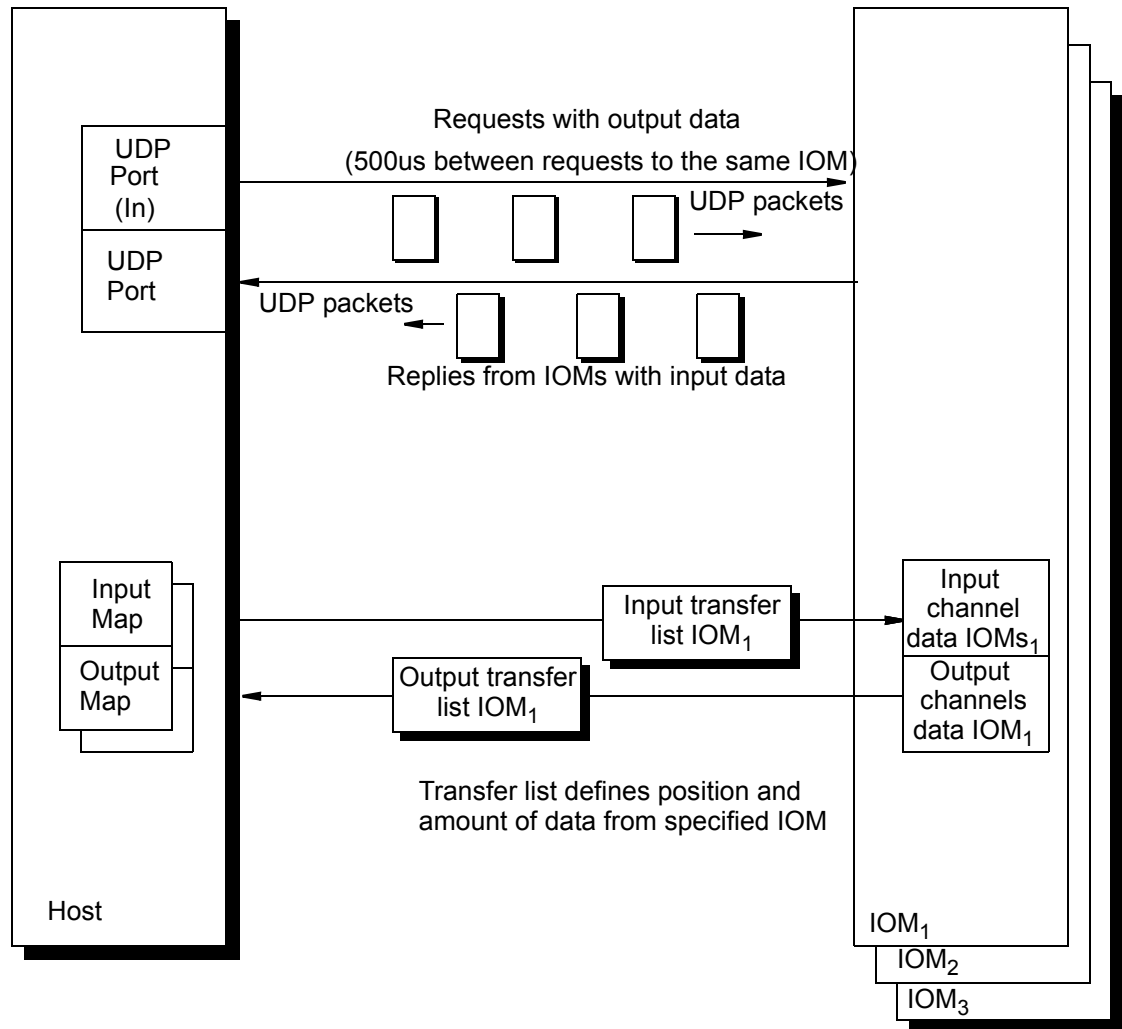


Figure 6-4. DMap Operation

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 they have to address different I/O boards (devices).

The maximum size of a DMap is limited to a maximum single packet size – 510 bytes. DMap allows representing 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.



Setting up a DMap is a multi-stage operation, which is completely automated in the DQE. The user is not required to set up module configuration, clocks, and a channel list. Instead, the DQE selects parameters that are best suited for the requested operation.

After you specify all channels you would like to see participating in data exchange, the DQE finalizes them. It parses the transfer list and sorts entries on a module/subsystem basis. After that, the DQE sets up configuration, acquisition rate, and channel list for every I/O board involved. Finally, the DQE sets up the transfer list for the IOM involved. Every transfer list within a host has a unique ID (DMap ID.) The DQE calculates addresses of data points for each entry into the transfer list.

The IOM converts the transfer list into a table of addresses associated with the DMap ID. After the DQE starts the DMap operation, it takes data from the output DMap and sends `DQCMD_WRRD` packets on a periodic basis. When the IOM receives a `DQCMD_WRRD` packet, it starts processing it, based on the received DMap ID. The IOM takes transferred data sequentially and writes it according to the output physical address table created earlier. After that, it reads data from the input physical address table and stores it into the reply packet. The DQE mirrors the IOM operation and stores data into the input DMap.

In addition, the DQE calls a conversion routine for each point of data to convert it from the raw representation into real-world values for both directions.

6.4 Variable-size Data Mapping Functions (VMap)

VMap is a protocol developed for control applications in which the ability to get immediate realtime data is equally important as receiving a continuous flow of the data.

VMap provides a realtime vehicle for messaging devices at the expense of restricting the ability to recover lost packets and shifting the decision whether or not to recover the lost packet to the user application. 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, DQ Engine 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.

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.



The user application should perform the following operations to ensure proper functionality of VMap mode:

1. Start DQ Engine and open communication with the IOM involved in operation.
2. Create VMap using **DqVmapCreate()**.
3. Add channels using **DqVmapAddEntries()**.
4. Call **DqVmapInitOps()** and **DqeEnable()** for the I/O boards included in the data exchange.
5. Fill out the output packet: first, amount of data to be sent for each channel, then the data at offsets corresponding with the specified size of data, and then specify how many bytes to receive for each channel. The size of header and data cannot exceed the packet size.
6. Write DQ_VMAP_READY flag into the flags field. Proceed with other VMaps handled by the application.
7. On the next DQ Engine tick, VMap callback will find the DQ_VMAP_READY flag set.
8. Send the packet to the IOM and then clear this flag and set the DQ_VMAP_PROCESSED flag. Since DQ_VMAP_READY and DQ_VMAP_PROCESSED are mutually exclusive, no critical section is required.
9. Wait on **DqeWaitForEvent()** for a little bit longer than packet turnaround time (and DQEngine tick time). If **eDataAvailable** is set, the input packet is available.
10. Make sure the DQ_VMAP_READY flag is set in the input buffer packet by the DQEngine.
11. Read data from the incoming packet according to the header. If FIFO information option is selected, the packet header will also contain data about how many samples are still waiting in the input FIFO and how much room is available in the output FIFO.
12. Clear the DQ_VMAP_READY flag and write the DQ_VMAP_PROCESSED flag to tell DQEngine that the buffer may be used over again.

If **ePacketLost** occurs, the user application has to make the choice of trying to repeat the lost packet or continue service as usual (i.e., send/receive the latest data).

If the user application needs to preserve continuity of the data, it should not change current data in the output buffer, but instead set the DQ_VMAP_REREQUEST flag. This flag tells DQEngine that the packet needs to be transmitted again with the same packet number.

The IOM stores the latest processed packet number (dqCounter from DQPKT) for each VMap handle. If the packet has the same number as a recently processed one, the IOM does not take any action besides repeating the reply to the last packet. In other words, no new data will be written to the output FIFO because this packet was already processed but became lost in transmission from the IOM to the host. If the IOM never received the packet with this number, the transmission error occurred while transmitting from the host to the IOM, so the packet is processed in the regular way. Data from the packet will be written to the output FIFOs and the new packet will be formed of the data from the input FIFOs.



If the user application decides that it does not need the lost packet, it should continue running as described in Step 5.

To finish VMap operation, disable operation with **DqeEnable()** call and call **DqVmapDestroy()** to free up resources.

6.5 Message Mode

With messaging devices (serial, CAN, ARINC interfaces), the data is a stream of bytes logically divided into frames, messages, strings, etc. There are two distinct features of messaging devices that make use of the DMap protocol inefficient, if not impossible, for handling messages:

1. The data is a stream and 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 and one cannot predict when and how much data will become available, and whether or not receiving/transmitting errors may exist on the bus.

Messaging layers are supported by the Msg protocol, which shares the same buffering mechanism as the ACB protocol. Inherently, 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

Message mode operates in much the same way as ACB mode. The IOM must have an I/O board installed that supports a messaging protocol, such as a CAN-503, an SL-501, or an ARINC-429 board. When messages are received by this board, they are stored in a FIFO. As with the streaming version of ACB mode, when the device is in Operation mode, a messaging board will send 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 then signals an event to alert the client program. The client program can then retrieve the messages and process them as desired.

There is also a Sending Message Queue on the host side, into which the client program can insert outgoing messages. The reader thread takes these messages from the queue and sends them to the IOM. The IOM will then transmit the messages on the network interface of the I/O board.

IOM / Host Data Transfer

When the messaging board receives a message, the message is stored on the board's FIFO. When the device (I/O board) is running in Operation mode, the board checks the FIFO at regular intervals, and sends any unsent messages to the host. The DQPKT in which a message is sent has the following field values:

```
dqProlog0xBABAFACA (standard)
dqTStampN/A
dqCounterSequential counter value
dqCommand0x1154 (RDFIFO reply)
rqIdSticky request ID from host
dqDataDQFIFO structure, whose data field contains one or more messages
```

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 **Figure 6-5** for an illustration.

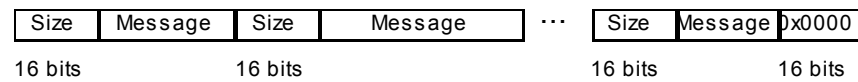


Figure 6-5. Data Field of a RDFIFO Packet Containing Messages

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 with a data field that holds one or more messages stored the same way.

The format of each message block is specific to the type of I/O board.

6.6 IOM Operating Modes

Once you have started up an IOM or a device, it can run on in any of four main modes of operation:

- Initialization
- Configuration
- Operation
- Shutdown

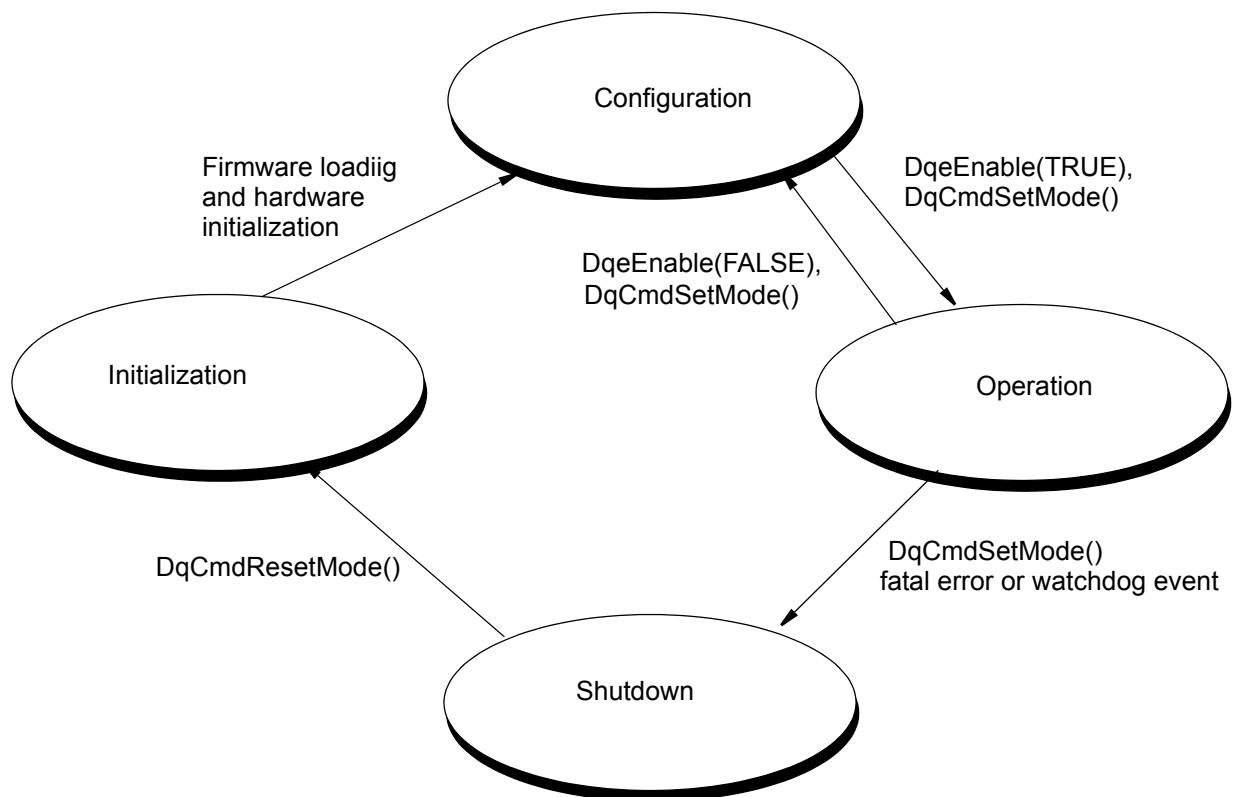


Figure 6-6. Modes of Operation



The host software controls the switching from state to state for each IOM by means of API function calls:

In each of the modes, it's important to understand what happens both at the host computer and at the IOM.

6.6.1 Initialization Mode

Host Behavior

- Does nothing

IOM Behavior

Upon power-up, each IOM:

- Copies user firmware into RAM and starts execution.
- Retrieves data from the parameter sector of flash memory and performs initialization accordingly.
- Initializes Fast Ethernet controller and TCP/IP stack.
- Initializes chip selects and creates memory map.
- Finds all devices attached to the bus (up to sixteen), assigns addresses and interrupts.
- Switches all devices into initialization mode by calling the proper entry point of each device driver.
- Reads EEPROM data and performs an initialization sequence. It resets each device and programs the states of the devices stored in EEPROM. It also programs the Transfer/Channel Lists. The driver also allocates all required memory structures and buffers and attaches them to the device object allocated by the initialization routine.
- Calls the driver entry function to switch every device into Configuration mode. In this mode, device drivers write calibration values into the digital calibration subsystem.
- Checks if the <autorun> field is 1. If it is 1, the firmware switches the device into Operation Mode. Otherwise, it switches to Configuration Mode.

The IOM keeps initialization data in the PARAM sector of the Flash memory (Flash device sector SA1) and EEPROM of every I/O board.

6.6.2 Configuration Mode

Configuration mode is designed to perform synchronous operations and to set up parameters for asynchronous operations. In this mode, you can set up operational parameters and the Channel List.

Host Behavior

The host can send and receive single commands.

IOM Behavior

Each IOM executes each DaqBIOS command upon its arrival. At this point, an IOM doesn't perform continuous acquisition or prepare data for I/O. In this mode, the IOM is capable of performing single-scan reads and writes. The first of those scans reads (or writes) programs to the channel list of the I/O board for one scan acquisition and starts low-rate data acquisition. All access to the data is organized using the `DQCMD_IOCTL` command, which resolves into



DqAdvxxxRead() and DqAdvxxxWrite() at the host level, where xxx is the model number of the I/O board.

In configuration mode, you can overwrite the current default parameters of configuration, clock settings, channel/transfer lists stored in EEPROM. All configuration-changing commands apply to the current control set of a board and do not affect EEPROM memory. This current control set of parameters is used to program the IOM when the host switches it into operation mode.

6.6.3 Operation Mode

Host Behavior

If DQE is running, the host continuously sends and/or receives a stream of data and controls data flow.

IOM Behavior

Each IOM performs continuous acquisition as defined in the transfer/channel lists. Once the IOM is switched into operating mode, it waits for a trigger to start operation. In the case of streaming operations, a trigger can be either an external event or a software command. In the case of data mapping, the IOM starts I/O immediately after switching into operating mode.

6.6.4 Shutdown Mode

Host Behavior

In shutdown mode, the host sends a shutdown request to all IOM modules under its control. Another way to enter shutdown mode is to program a watchdog clock on an IOM and allow it to expire. A watchdog timer clock resets every time an IOM doesn't receive packets from the host for a specified period of time. Note that when the IOM module is set up to perform input streaming, operation data is sent from the IOM side and the host intervenes only on an error. In this situation, the user should enable heartbeat operation before engaging the watchdog.

In shutdown mode, it is most important for an output device to switch its output into a predictable state (voltage level, tri-stated, etc.)

IOM Behavior

When an IOM receives a request for Shutdown mode, it executes a programmed sequence. It brings all I/Os to a safe state, which is stored in a shutdown parameter area of the module EEPROM.

An IOM also can switch itself into Shutdown state on the following events:

- The watchdog timer fires
- Communication with the host is lost and cannot be resumed in a specified period of time

One of the ways to ensure fail-safe operations is to set up <autorun> in the <fwct> parameter to 1. Then, when the watchdog fires, it resets the IOM completely. Then the IOM switches, via initialization and configuration modes, into operation mode without intervention from the host. In this circumstance, if for example, the host was engaged in data mapping operations and one IOM fails, communication will be lost during the time of booting up this IOM and switching back into operating mode. Once in operation mode, the IOM becomes accessible again with the parameters stored in EEPROM operation mode section.



6.7 Heartbeat Feature

The heartbeat feature allows the DQE to keep track of which IOMs are available on the network. To achieve this goal, every IOM entry has a special `ready` field. If this field is `TRUE`, the IOM is alive.

6.7.1 Heartbeat and Safe States

An IOM can have a watchdog timer set up to reset the PowerDNR system (hardware reset) if the idle thread doesn't clear this counter periodically. There are two ways to set it up:

- Use the command from the serial interface: "time watchdog N", where N is time in ms between counter resets. If this time is 0, the watchdog timer is disabled
- Use `DQCMD_SETPRM` command with `IOMODE_NAMEDPRM` mode -- parameter name is `DQPRM_WATCHDOG`

An IOM can switch itself into shutdown mode or reset the PowerDNR system upon loss of communication. The host side is responsible for setting this mode

- Use `DQCMD_SETPRM` command with `IOMODE_NAMEDPRM` mode -- parameter name is `DQPRM_COMMLOST` -- to set up:

```
// Number of milliseconds before switching to shutdown mode
// Reset flag to reset firmware and switch to initialization and then
// configuration mode upon reaching shutdown mode
// Write 0 to switch this mode off
```

When this mode is active, the IOM expects to receive any valid packet from the host side every N milliseconds. If there is no communication within this timeframe, the IOM sets up safe values on its outputs.

The host has a special sticky entry (Heartbeat Entry, see below) with the `DQCMD_ECHO` command in the CQ scheduled to be sent before the communication timeout expires. The command will not always be sent to the IOM because the receiving thread will reset the time-to-send field of this entry each time a packet arrives from an IOM. This mechanism is especially useful for output streaming because the IOM doesn't send any replies if there are no errors in the stream.

6.7.2 Heartbeat and Moving Token Mechanism

In each command queue, there can be a special entry – the Heartbeat Entry. It is given a special treatment by both receiving and sending thread.

6.7.3 Heartbeat Processing in Sending Thread

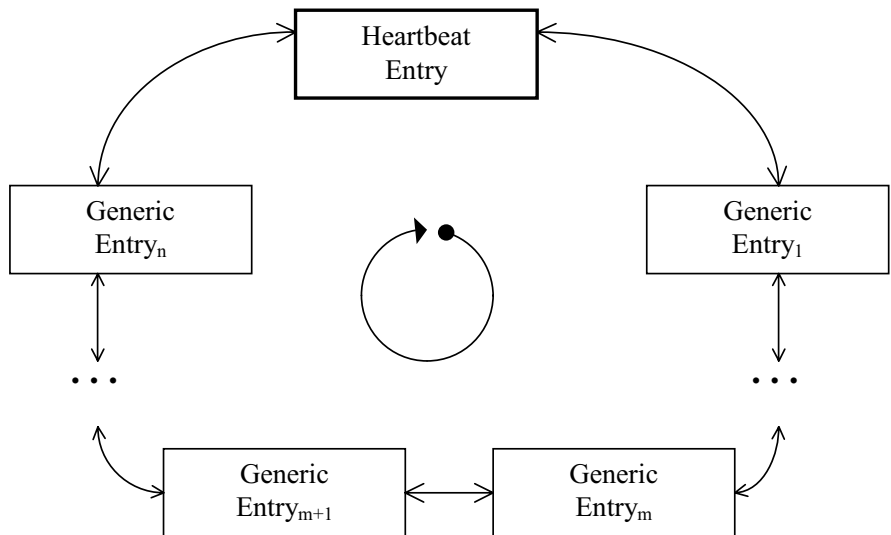
A sending thread always starts processing the Command Queue (CQ) from the Heartbeat Entry. If it ever reduces time-before-timeout to zero, the IOM will be marked "unavailable" and no other commands except the Heartbeat Entry's `DQCMD_ECHO` will be sent.

In normal operation, Heartbeat Entry will also serve as a token, marking the place where we last stopped processing the CQ. After going through the whole CQ, if the sending thread has sent any command this cycle, the sending thread moves the Heartbeat Entry from its current position directly after the corresponding entry (the one associated with the command just sent). This ensures rotation of all commands in the CQ.

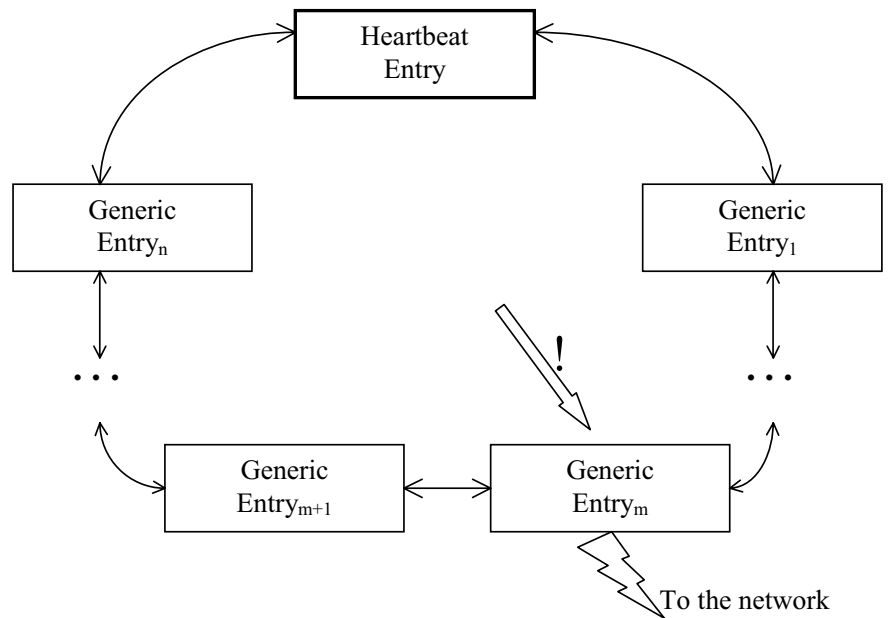


The mechanism works as follows:

STEP 1: Sending thread starts going through the CQ from Heartbeat Entry:



STEP 2: Sending thread finds a ready-to-send entry and sends the packet:



Chapter 7 DaqBIOS Protocol

7.1 DaqBIOS Packet Structure

The DaqBIOS (DQ) protocol relies on the Ethernet protocol for exchange of data between IOM and host. Current implementation of the IOM firmware allows transferring DaqBIOS packets over raw Ethernet packets or over UDP packets. Library implementation under Microsoft Windows, however, does not offer the 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 a simple concept of acknowledging every packet sent from host to IOM.

The DaqBIOS packet header has the 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 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, continues up to 65535, and 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 doesn't 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 a reuse of the previous code
** in the 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 by 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 figure depicts how the host and the IOM exchange packets under the DaqBIOS protocol:

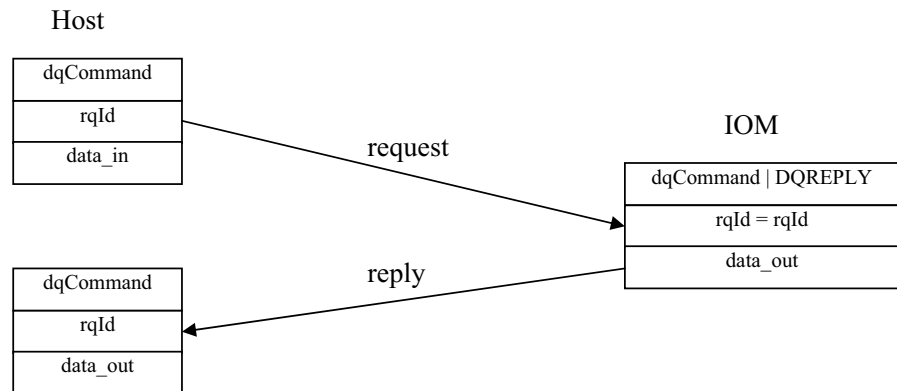


Figure 7-3. Host/IOM Packet Exchange in DaqBIOS Protocol

7.2 DaqBIOS Protocol Versions

To determine what version of the DaqBIOS protocol the DNR-12 system supports, the host should send a command with `dqProlog` set to `0xBABAFAC2`. The IOM will reply with the proper prolog and 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 IOM 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 is represented in big-endian format. Data on the PC platform is rendered in little-endian format. Thus, to ensure proper data representation, you should convert data from network to host format and vice versa.

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 operating systems (QNX, for example) or general-purpose operating systems 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. In 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. 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 module capabilities information. The time for pending command execution varies and the user should adjust the timeout appropriately before calling these commands.

7.3.2 DaqBIOS & Network Security An IOM may be connected to the Internet, posing virtually no risk to the network it is hosted on. Several features make the IOMs virtually invulnerable to external attack, in descending order:

1. The IOM 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 IOM 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 the internal state of the DAQ system requires a user password to be supplied. The password is stored in the encoded NVRAM area of RTC chip. Any command that changes non-volatile memory requires the super-user password. Password is supplied over DQ protocol.
4. To prevent disruption of the experiment, the IOM 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 system, the IOM locks on to the host's IP/port pair and listens for commands only from the locked host – until the host unlocks/releases the IOM. Other PCs can only request system configuration and status requests (e.g. IOM_25431 with an AI-201 module in slot 0 is currently in Locked state).

Finally, note that the IOMs have 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 linked with a user application. 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 own purposes. Various user applications can use DQE simultaneously. Every process gets its own copy of DQE. DQE implements interlock mechanisms, preventing use of a single IOM by two processes and a single module in exclusive modes.

DQE simplifies IOM programming and shifts data contingency and buffering responsibility from a user application to the library.

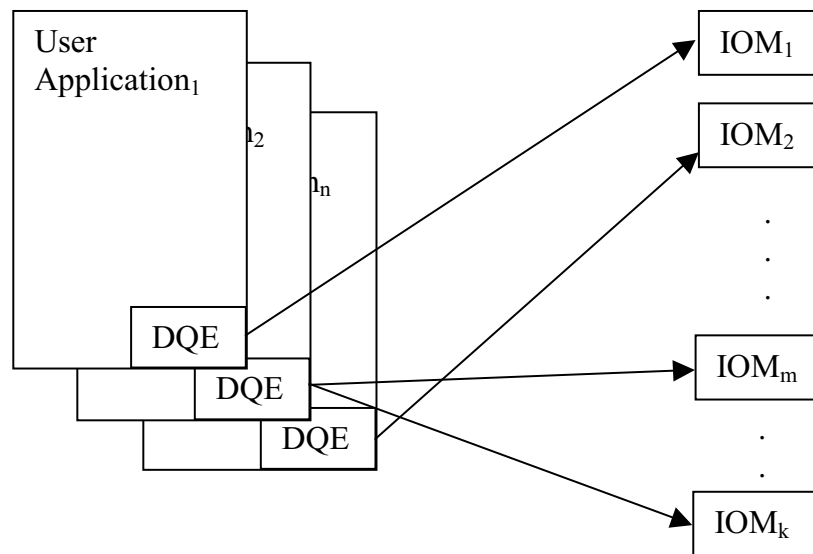


Figure 8-1. User Application/DQE/IOM Interaction.

In **Figure 8-1**, note that one user application may interact with more than one IOM. The converse is not true.

8.1 Basic Architecture

The DaqBIOS Engine consists of the following parts:

- **Sending Thread/Periodic Task** (multimedia timer callback under Windows)

There is a single sending thread in every DQE. 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 them as “waiting for response” so that they aren’t sent the next time. (Refer also to the Command Queue entry below.)



- **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 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 any 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 modules 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 from or sent to 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 `DqRecvPkt()` calls or DQE callbacks, as 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 routines depending on the module 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 an IOM.

8.2 Threads and Functions

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 Conversion

The reader and writer threads call a conversion routine that converts data from the raw format represented in the ring buffer into floating point representations 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 we dedicated a separate chapter for real time operation is that writing real-time code can be done more efficiently without using the DQE. That's why we will talk about programming streaming and data mapping operations at low-level in this section. (Refer to "Synchronous Mode Programming" on page 109.)

9.1 Real-time Data Mapping (RtDmap) Functions

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

Operation is as follows:

At each clock tick, the IOM's 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 that it synchronizes periodically with the IOM's version of the DMap.

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. 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 low-level API (**DqDmap***** functions) uses the DqEngine to refresh the DMap at a given rate and to retry a DMap refresh request if, for some reason, a packet is lost.

The DqEngine is necessary on desktop-oriented operating systems to ensure that the DMap is refreshed periodically, but is overkill on real-time operating systems.

The following is a list of the real time data mapping functions, with short descriptions:

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 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.
DqRtDmapGetOutputMap	Gets 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.



Table 9-1 RtDmap API Functions (Cont.)

Function	Description
DqRtDmapReadScaledData	Reads and scales data stored in the input map for the specified device. Note: The data read is the data transferred by the last call to DqRtDmapRefresh(). 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 DqRtDmapRefresh(). This function should only be used with devices that acquire 16-bit wide digital data such as the DIO-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 DqRtDmapRefresh(). 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 DqRtDmapRefresh(). 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 DqRtDmapRefresh(). 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 DqRtDmapRefresh(). 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 DqRtDmapCreate.
DqRtDmapStop	This function stops operation and the IOM stops updating its internal representation of the data map.



Table 9-1 RtDMap API Functions (Cont.)

Function	Description
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 DqRtDMapInit(). 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 DqRtDMapInit(). 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 DqRtDMapInit(). 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 allocated by the DMAP operation on the specified IOM.



9.2 Real Time Variable-size Data Mapping (RtVmap)

This feature is similar to RealTime DMap operation (see “Real-time Data Mapping (RtDmap) Functions” on page 100) 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.

The following table is a list of the real time variable data mapping functions, with short descriptions of each.



Table 9-2RtVmap API Functions

Function	Description
DqRtVmapInit	Initialize the specified IOM to operate in VMap mode at the specified refresh rate. Note: The term <code><refresh rate></code> has only informational meaning for now.
DqRtVmapAddChannel	This function adds a channel to <code><vmapid></code> 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. Channel in <code><cl></code> should be defined in the standard way including channel number, gain, differential, and timestamp flags. Configuration <code><flags></code> 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 <code><flags></code> 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. <code><clSize></code> specifies the maximum number of array entries. The Output VMap buffer, which transfers data from host to IOM, has the structure shown in Table 9-3 on page 107. The total length of the buffer cannot exceed the size available in the UDP packet minus the combined size of the DQPKT and DQRRD headers. 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. 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.



Table 9-2RtVmap API Functions (Cont.)

Function	Description
DqRtVmapAddChannel (cont.)	The Input VMap buffer, which transfers data from IOM to host, has the structure shown in Table 9-4 on page 107. 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. The header size cannot be changed after DqRtVmapStart() is called. In other words, after a channel is added using DqRtVmapAddChannel(), the header size increases by one in the output packet and by one or two (if DQ_VMAP_FIFO_STATUS 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. Note: Each call to DqRtVmapAddChannel() 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 a transfer list index equal to 9.
DqRtVmapStart	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. If clocked devices (AIn/AOut) are used, the function programs devices at the rate specified in DqRtDmapInit.
DqRtVmapStop	This function stops operation and the IOM stops updating its internal representation of the data map.
DqRtVmapClose	This function destroys the <vmapid> VMap.
DqRtVmapRefresh	This function refreshes the host version of the map by downloading the IOM map. Use the DQ_VMAP_REREQUEST flag if you want to re-request the failed transaction instead of performing a new one. In such case, the dqCounter in the DQPKT 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. Note: The IOM automatically refreshes its version of the data map at the rate specified in DqRtVMapInit(). This function should be called periodically (a real time OS is required) to synchronize the host and IOM data maps).



Table 9-2RtVmap API Functions (Cont.)

Function	Description
DqRtVmapRefreshOutputs	This function refreshes the host version of the map by downloading the IOM map. Use DQ_VMAP_REREQUEST flag if you want to re-request the failed transaction instead of performing a new one. Note: This function needs to be called periodically (real-time OS is required) to synchronize host and IOM data.
DqRtVmapRefreshInputs	This function refreshes the host version of the map by downloading the IOM map. Note: This function needs to be called periodically (a real-time OS is necessary) to synchronize the host and IOM data maps.
DqRtVmapGetInputPtr	This function gets the pointer to the beginning of the input data allocated for the specified entry. Note: This function can be called only after packet is received.
DqRtVmapGetOutputPtr	This function gets the pointer to the beginning of the output data allocated for the specified entry. Note: This function can be called only after transmission size for all channels is written.
DqRtVmapGetInputMap	Get pointer to the beginning of the input data map allocated for the specified device. 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.
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() .



Table 9-2RtVmap API Functions (Cont.)

Function	Description
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 9-3Output VMap Buffer

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

Table 9-4Input VMap Buffer

	Flags (uint16)
Number of bytes retrieved from Ch0 (uint16)	Number of bytes remaining in Ch0 (uint16, optional)
...	...
Number of bytes retrieved from ChN (uint16)	Number of bytes remaining in ChN (uint16, optional)
Number of bytes written to Ch0 (uint16, optional)	Number of bytes that can be written to Ch0 (uint16, optional)
...	...
Number of bytes written to ChN (uint16, optional)	Number of bytes that can be written to ChN (uint16, optional)



Table 9-4*Input VMap Buffer (Cont.)*

	Flags (uint16)
Data from Ch0 (of specified retrieved size)	
...	
Data from ChN (of specified retrieved size)	

9.2.1 Real-time Operation: in Receiving Thread

Each time a receiving thread receives a packet from an IOM, it resets the time-to-send and time-before-timeout fields in the Heartbeat Entry to the maximum and clears the waiting-for-answer flag. It also marks the IOM “available” in the IOM table. This ensures that the `DQCMD_ECHO` command, associated with the Heartbeat Entry, is sent out to the IOM only in case of prolonged silence, and even if it was sent and IOM responded in the meantime, there will be no alarm.



9.3 Synchronous Mode Programming

Programming in synchronous mode can be done with or without invoking the DQE. Most DaqBIOS commands include a request sent from the host and a reply sent back from the IOM to the requesting host.

When the library is used without the DQE, each function call sends a packet to the IOM and waits for the reply from it. Thus, the function does not return control to the calling process before the IOM sends a reply or before the timeout expires. All underlying function calls occur in the context of the user application process. The timeout in milliseconds is set in `DqOpenIOM()` or `DqSetTimeout()`. There is a separate timeout for each IOM.

When the DQE is running, a function call does not send a packet directly, but instead adds the packet into the command queue (CQ.) This entry receives a unique system-wide request ID. Then the function call relinquishes control to the OS on the wait function. It waits until either an event or a timeout is received.

On the next clock cycle, the sending thread finds the new entry in the command queue and sends the packet to the destination IOM. At the same time, it decreases the retry counter in the command queue entry.

When the receiving thread receives the packet with the same request ID, it deletes the command queue entry and copies the packet data into the buffer belonging to the waiting function call.

Every time the sending thread receives a clock cycle, it decreases the timeout value in the command queue entry. When a timeout occurs, it sends the same packet again and decreases the retry counter. This way, every synchronous command is repeated several times before the sending thread releases the waiting user call with a timeout.

If the IOM doesn't reply, the total timeout equals the reply timeout multiplied by the number of retries multiplied by the DQE clock period. `DqSetTimeout()` can change the wait time on the fly for all IOMs addressed by the DQE. The user, however, cannot change the timeout time for each IOM.



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:

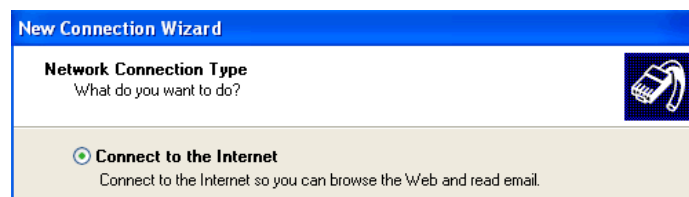
- STEP 1:** From the *Start* menu, select *Control Panel*, and click *Printers and Other Hardware*.
- STEP 2:** From the menu on the left, click *Add Hardware* and follow the on-screen instructions.

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*.

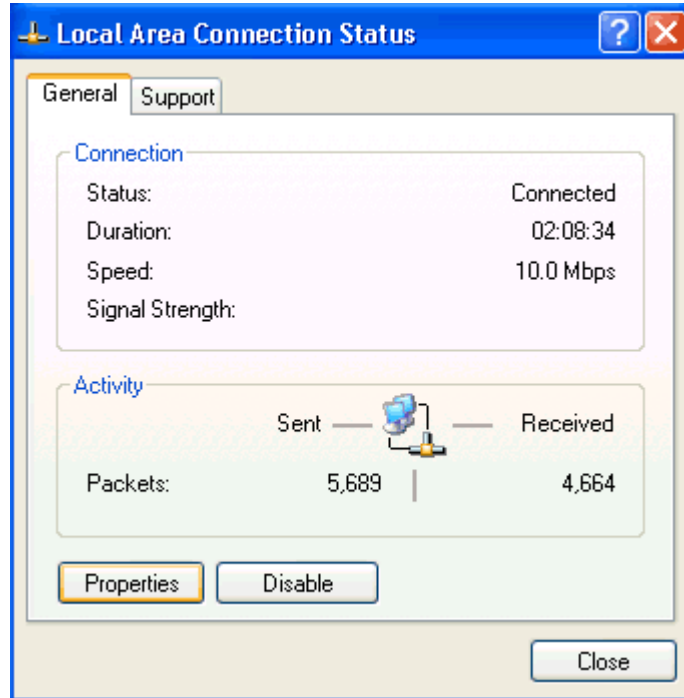


STEP 10: Select *Connect Using a Broadband Connection that is always on* and click *Next*.

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.

A.2 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.

A.3 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.

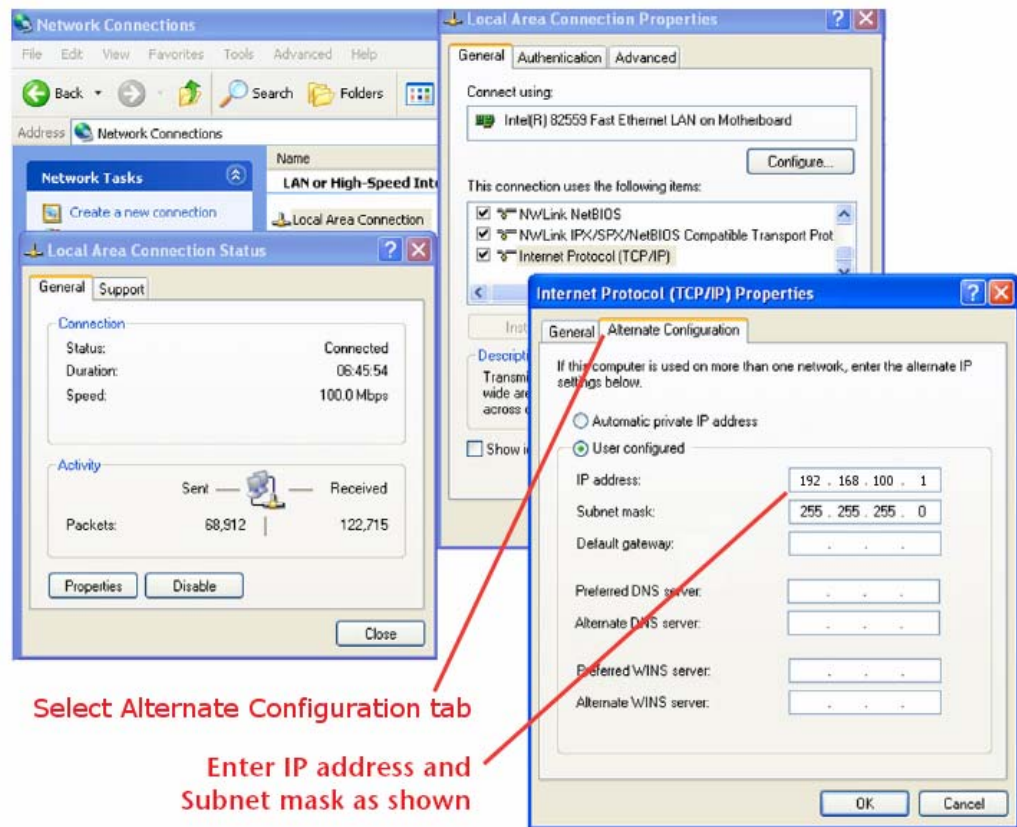
Windows XP users have the ability to configure a second IP address setting under the Control Panel that will allow Windows to pick the correct computer IP setting, based on the device that it finds connected to the Ethernet port. Under this configuration, your primary IP setting is configured for *Obtain 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 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.4 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:



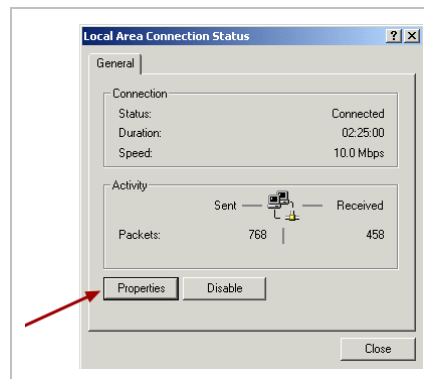
A. Set Up Your Ethernet Card (NIC)

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.

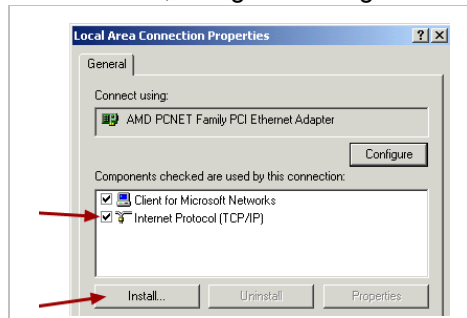
- STEP 1:** From the *Start* menu, select *Settings* and then select *Network and Dial-up Connections*.
- STEP 2:** If you see a *Local Area Connection* icon, your Ethernet card has been detected and installed, skip ahead to the section *Configure TCP/IP*. If you do not see this icon, proceed to step 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. Installing 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*:



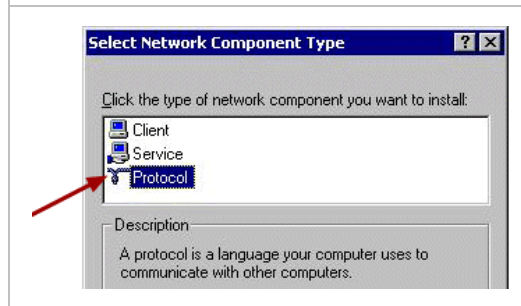
- STEP 4:** If Internet Protocol (TCP/IP) is listed, make sure the box next to it contains a check mark, and go to *Configure TCP/IP*.



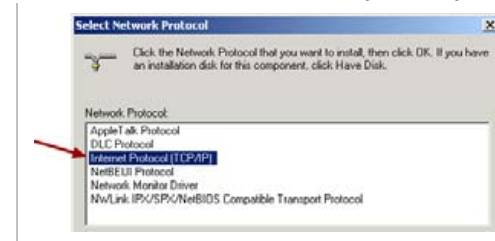
- STEP 5:** If Internet Protocol (TCP/IP) is not listed, click on *Install*.



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



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



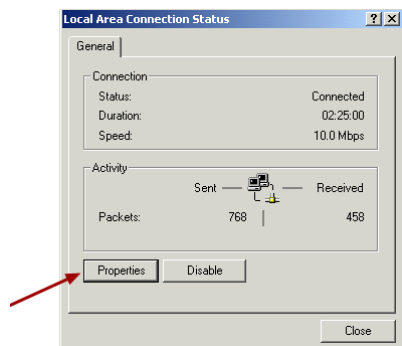
STEP 8: Make sure the box beside *Internet Protocol (TCP/IP)* contains a check mark, and proceed to the next section, *Configure TCP/IP*.

C. Configure TCP/IP

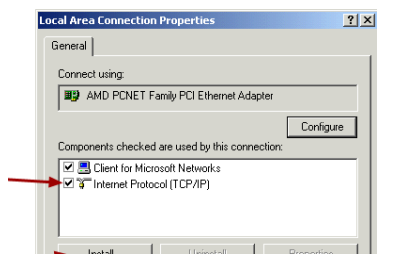
STEP 1: From the Start menu, select Settings and then select Network and Dial-up Connections.

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:



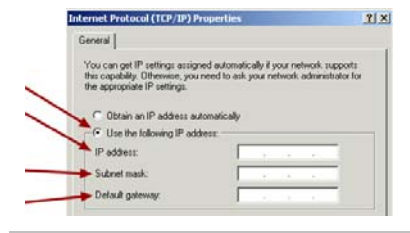
STEP 4: Click once on Internet Protocol (TCP/IP). Then click Properties.



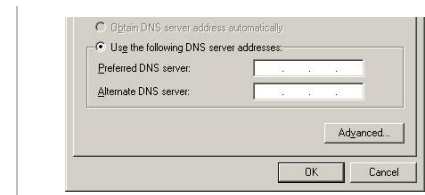
STEP 5: Select *Use the following IP address*, and type 192.168.100.1

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.5 Configuring a Second Ethernet Card Under Windows NT

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.

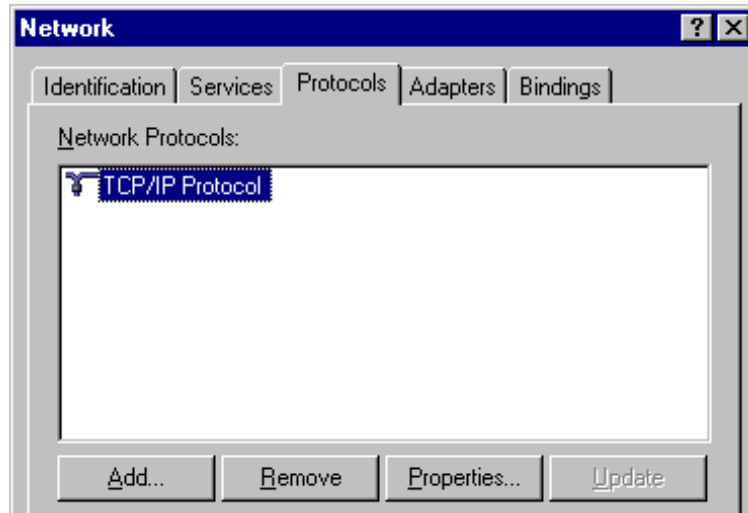
If you obtained an Ethernet interface after Windows NT was already on your computer, do the following:

- 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.



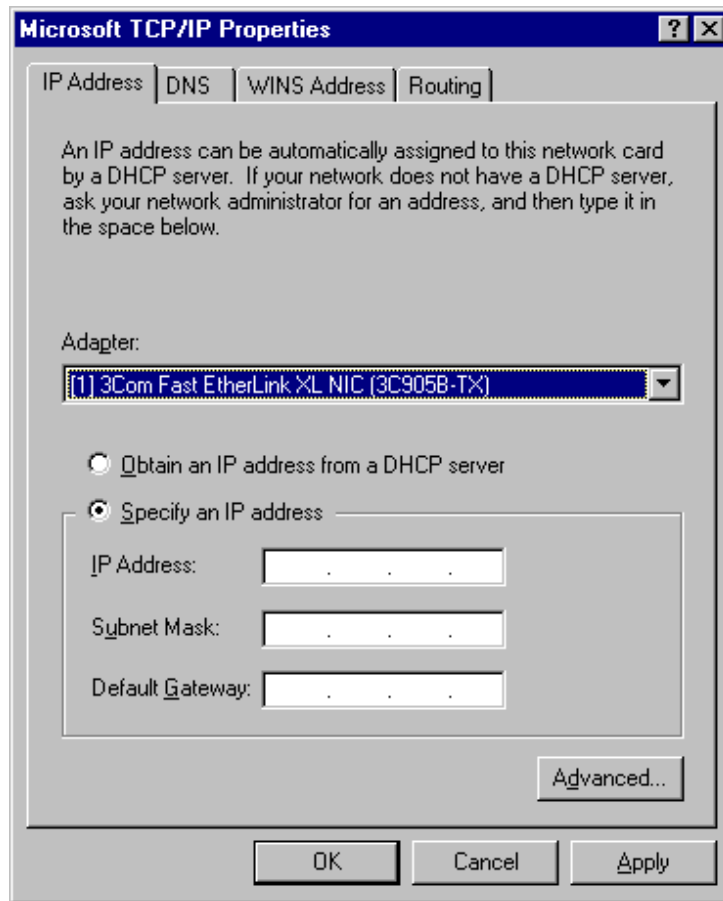
B. Install and Configure TCP/IP

- 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.6 Configuring a Second Ethernet Card Under Windows 95/98/SE/ME

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.

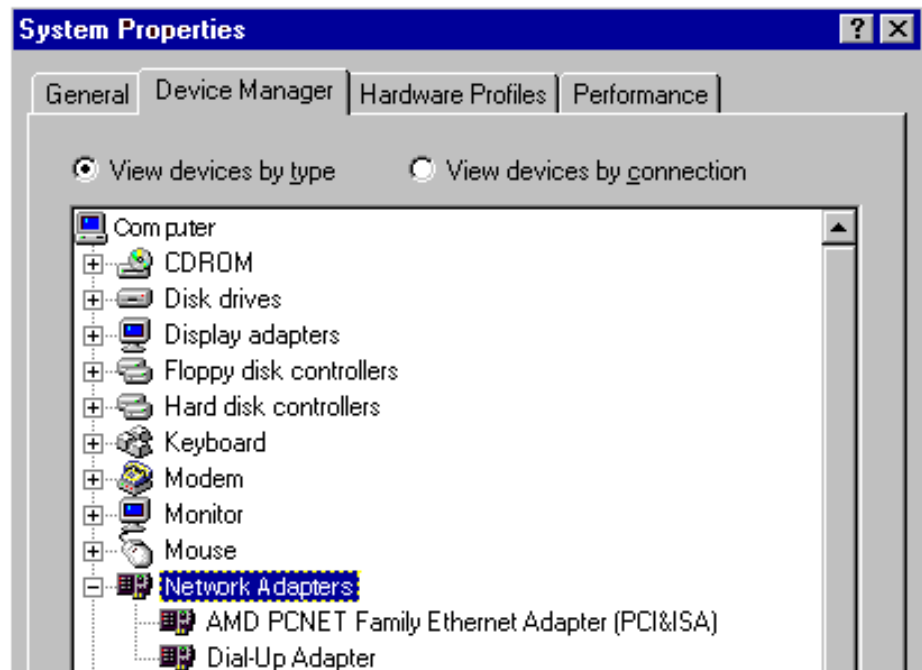
If you obtained an Ethernet interface after Windows 95/98/ME was already on your computer, then do the following:

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.



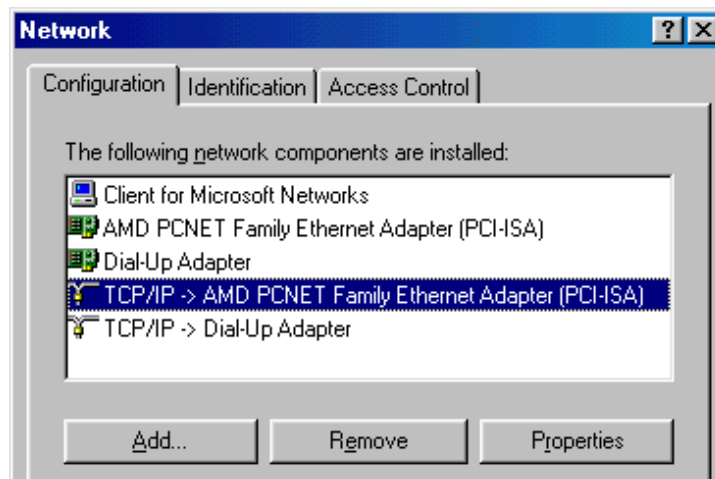
- STEP 4:** If an entry for your second Ethernet card appears here, you probably do not need to run any software included with your card, but keep the software handy just in case you need it later to resolve a problem.
- STEP 5:** Note the name of your second Ethernet card.
- STEP 6:** Close the System 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.

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.

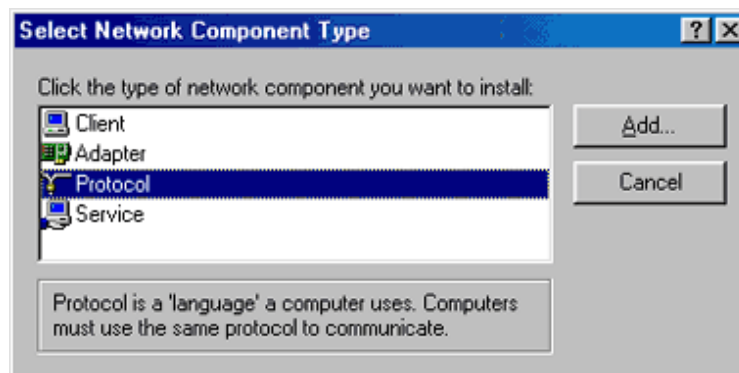
STEP 4: If you see IPX/SPX-compatible Protocol or NetBEUI in the list, select it, then click the Remove button to delete it. These protocols are used by some networked applications, especially games, but they may interfere with your Ethernet connection.

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.

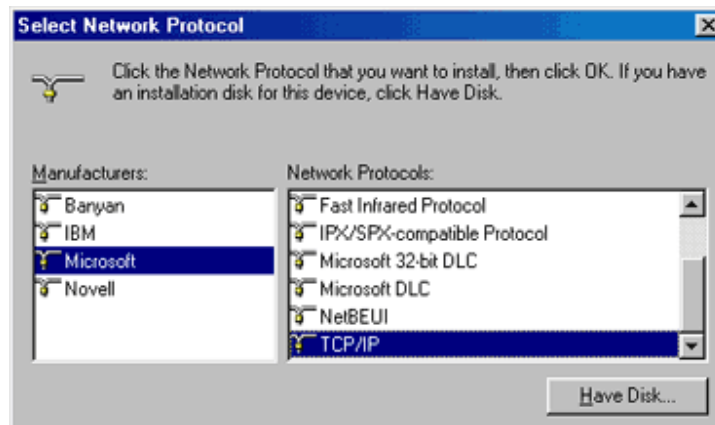
STEP 6: In the Network control panel, click the Add... button.

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.

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.

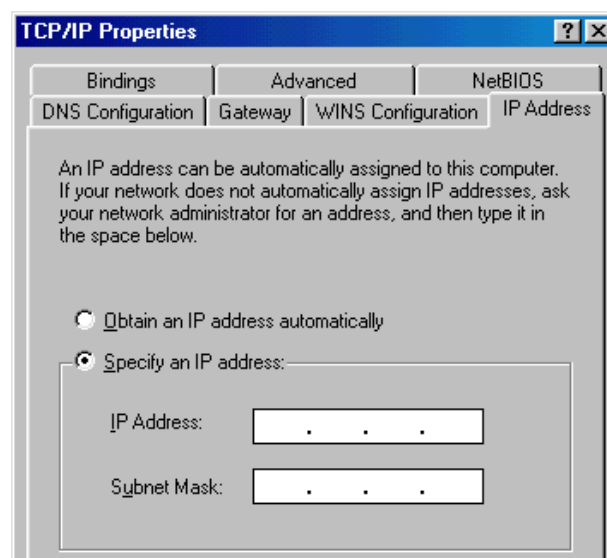
STEP 2: In the box labeled The following network components are installed, select TCP/IP. TCP/IP is listed at least twice, so choose the one followed by the name of your second Ethernet card (do not choose TCP/IP -> Dial-up Adapter).

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.

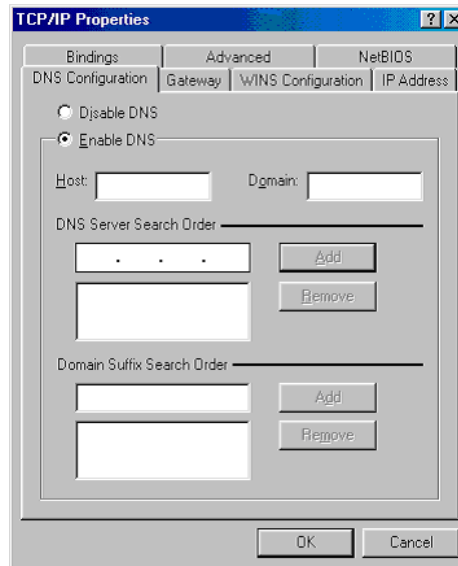


STEP 7: Click on the DNS Configuration tab.

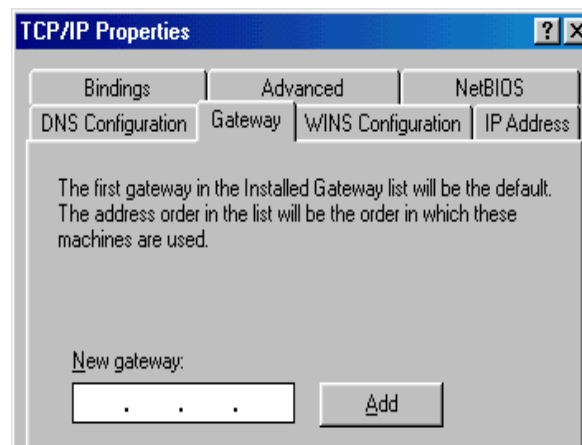


STEP 8: Select Enable DNS.

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.

Appendix B

Field Replacement of Fuses on DNA and DNR Boards

Some boards used in UEI DAQ I/O systems require field replacement of fuses when unexpected overloads occur. Locations of these fuses are shown in **Figure B-1** through **Figure B-3**. Part numbers for the replacement fuses are listed **Table B-1**.

Table B-1. DNA/DNR Replacement Fuses

UEI Fuse ID (Board)	Rating	UEI Part No.	Description	Mfr.	Mfr P/N
F1	5A	925-5125	FUSE 5A 125V SLO SMD SILVER T/R	Littlefuse	0454005.MR
F2	5A	925-5125	FUSE 5A 125V SLO SMD SILVER T/R	Littlefuse	0454005.MR
F3 (DC)	5A	925-5125	FUSE 5A 125V SLO SMD SILVER T/R	Littlefuse	0454005.MR
F3 (1GB)	10A	925-1125	FUSE 10A 125V FAST NAN02 SMD	Littlefuse	0451010.MRL
F4	5A	925-5125	FUSE 5A 125V SLO SMD SILVER T/R	Littlefuse	0454005.MR

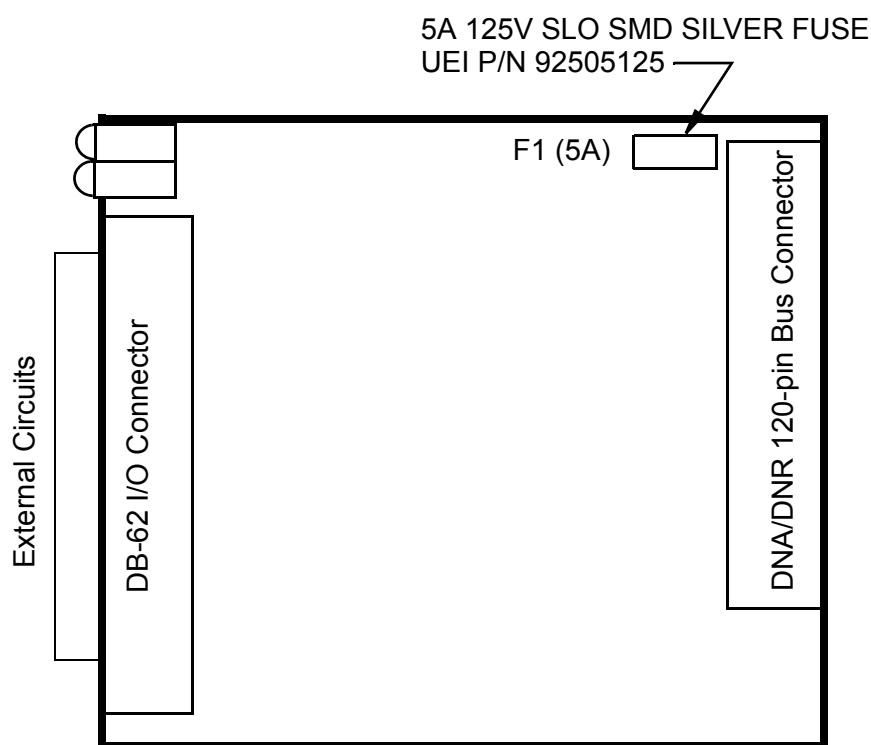


Figure B-1. Location of Fuse for PL-61x, PL-62x, and PL-63x Boards

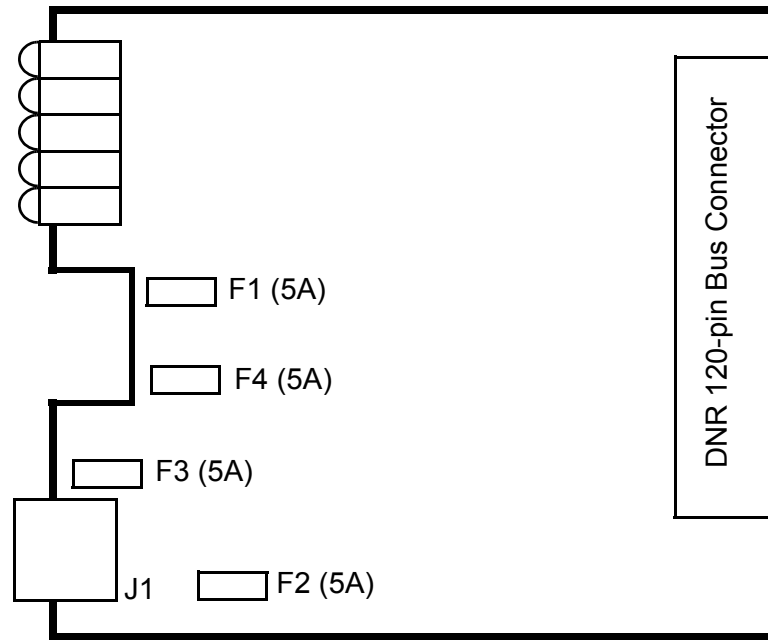


Figure B-2. Location of Fuses for DNR-POWER-DC Board

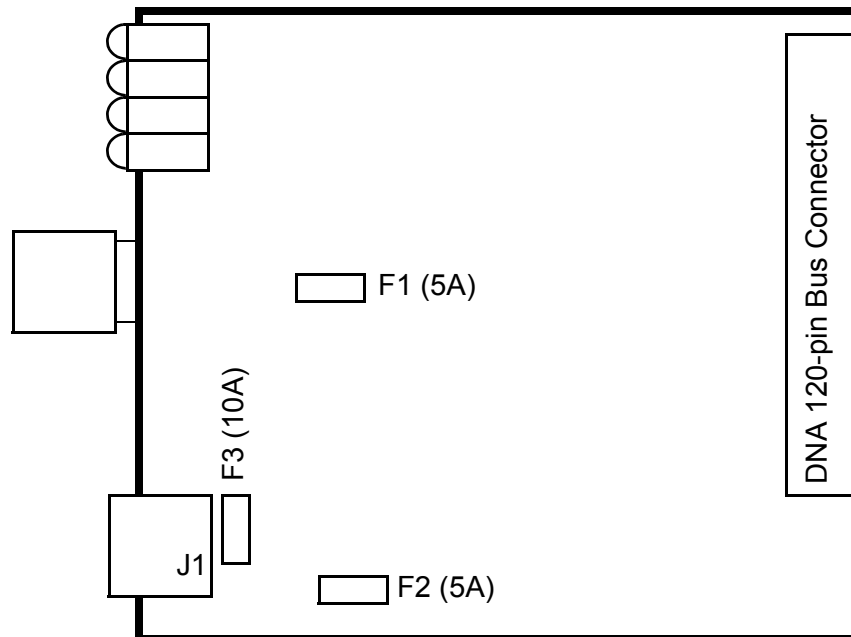


Figure B-3. Location of Fuses for DNA-POWER-1GB Board

Index

Symbols

"Show" Command 22

B

Boot-up 21

C

Channel List 70

Clock and Watchdog Access 69

Common Layer Interface 70

Configuration Flags 72

Configuring a Second Ethernet Card Under Windows 2000 113

Configuring a Second Ethernet Card Under Windows 95/98/SE/ME 118

Configuring a Second Ethernet Card Under Windows NT 116

Configuring a Second Ethernet Card Under XP 110

Conventions 2

D

DaqBIOS

- Advanced Circular Buffer 98

- Basic Architecture 97

- Buffer Control Block 98

- Command Queue 98

- Data Map 98

- Data Representation 95

- Immediate and Pending Commands 96

- IOM Table 98

- Network Security 96

- Packet Structure 93

- Receiving Thread 98

- Sending Thread/Periodic Task 97

- Soft and Hard Real-time 95

- Threads and Functions 98

- Versions 95

DaqBIOS Engine 97

Default IP Address 23

DHCP 27

DMap Structure and Function 83

DNR Core Module

- Device

- Architecture 60

DNR-CPU-1000 Core Module 59

DNR-POWER-DC 21

E

EEPROM User Area Access 73

F

Field Connections 35

Fuse Replacement 123

G

Gateway Mask 24

H

Heartbeat 91

Heartbeat Processing 91

Host / IOM Communication Modes 78

I

Improving Network Performance 25

Installing Software 19

IOM Data Retrieval and Conversion 99

L

Layer Signaling 75

M

Mapped Messaging (M3) 78

Messaging 78

Modifying I/O layers 37

Modifying the IP Address 23

Mounting 35

MTTTY Screen 22

N

Network Mask 24

O

Operating Modes 88

- Configuration 89

- Initialization 89

- Operation 90

- Shutdown 90

Organization of Manual 1

P

Packet Ring Buffer 81

PowerDNA Explorer 43

- Analog Input Layer Settings 55

- Analog Output Layer Settings 54

- Counter/Timer Layer Settings 56

- Device Tree 46

- DIO Layer Settings 49

- File Menu 43

- Help 46

- Layer Settings 48

- Main Window 43

- Menu Bar 43

- Network Menu 44
- Settings Panel 47
- Toolbar 46
- View Menu 46
- Programming Layer Functions 63

R

- Real Time Variable-size Data Mapping (RtVmap) 103
- Real-time Operation (RtDmap) 100
- Repairs 37
- Reset Button 34

S

- Self-Diagnostics 22
- Setting CM Parameters 64
- Setting Parameters Via Serial Interface 67
- Setup Program 20
- Software Install -- Linux 21
- Specifications 4
- Startup Sequence 63
- Support ii

- Support email
 - support@ueidaq.com ii
- Support FTP Site
 - ftp
 - //ftp.ueidaq.com ii
- Support Web Site
 - www.ueidaq.com ii
- Synchronous and Asynchronous Modes 79
- Synchronous Mode Programming 109
- System Front Panel Layout 21

T

- Terminal Emulation Program 21

U

- u_boot 34
- Updating Firmware 32
- Upgrades 37

W

- Windows
 - Registry 20