DNx-DIO-480
User Manual

Universal 32-Channel DIO Board
for the PowerDNA Cube and RACK Series Chassis

April 2024

PN Man-DNx-DIO-480

See the UEI website for complete terms and conditions of sale:
http://www.ueidaq.com/cms/terms-and-conditions

CE

Contacting United Electronic Industries

Mailing Address:
249 Vanderbilt Avenue
Norwood, MA 02062
U.S.A.

Shipping Address:
24 Morgan Drive
Norwood, MA 02062
U.S.A.

For a list of our distributors and partners in the US and around the world, please contact a member of our support team:

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:	uei.support@ametek.com
Website:	www.ueidaq.com
FTP Site:	ftp://ftp.ueidaq.com

Product Disclaimer:

WARNING!

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

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

Specifications in this document are subject to change without notice. Check with UEI for current status.

Table of Contents

List of Figures

List of Tables

Chapter 1 Introduction

This manual outlines the feature set and use of the DNx-DIO-480, a 32-channel digital IO board. Each of the 32 channels can be independently configured for input or output and are compatible with 3.3 V TTL signals and industrial signal levels up to 55 V.

The following sections are provided in this chapter:

- Organization of this Manual (Section 1.1)
- Manual Conventions (Section 1.2)
- Naming Conventions (Section 1.3)
- Related Resources (Section 1.4)
- Before You Begin (Section 1.5)
- DNx-DIO-480 Features (Section 1.6)
- Technical Specifications (Section 1.7)

1.1 Organization of this Manual

This DNx-DIO-480 User Manual is organized as follows:

- **Introduction**
  Chapter 1 summarizes the features and specifications of the DNx-DIO-480.
- **DIO Functional Description**
  Chapter 2 provides an overview of the industrial digital inputs and outputs and their respective operating modes. Also included in Chapter 2 is information on TTL digital input and output, Guardian Diagnostic reads, circuit breakers, and pinout information.
- **PowerDNA Explorer**
  Chapter 3 shows how to explore DNx-DIO-480 features through a GUI-based application.
- **Programming with the High-level API**
  Chapter 4 describes how to configure the DNx-DIO-480, read data, and write data with the Framework API.
- **Programming with the Low-level API**
  Chapter 5 provides an overview of using C code for configuring and using the DNx-DIO-480.
- **Accessories**
  Appendix A provides a list of accessories available for use with the DNx-DIO-480.

1.2 Manual Conventions

The following conventions are used throughout this manual:

Tips are designed to highlight quick ways to get the job done or to reveal good ideas you might not discover on your own.

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

NOTE: Notes alert you to important information.

Typeface	Description	Example
bold	field or button names	Click **Scan Network**
»	hierarchy to get to a specific menu item	**File » New**
`fixed`	source code to be entered verbatim	`session.CleanUp()`
<brackets>	placeholder for user-defined text	`pdna://<IP address>`
italics	path to a file or directory	*C:\Program Files*

1.3 Naming Conventions

The DNA-DIO-480, DNR-DIO-480, and DNF-DIO-480 board versions are compatible with the UEI Cube, RACKtangle, and FLATRACK chassis respectively. These boards are electronically identical and differ only in mounting hardware. The DNA version stacks in a Cube chassis, while the DNR and DNF versions plug into the backplane of a Rack chassis. Throughout this manual, the term DNx-DIO-480 refers to both Cube and Rack products.

1.4 Related Resources

This manual only covers functionality specific to the DNx-DIO-480. To get started with your Cube or Rack, please see the documentation included with the software installation. On Windows, these resources can be found from the desktop by clicking **Start » All Programs » UEI**

UEI's website includes other user resources such as application notes, FAQs, tutorials, and videos. In particular, the glossary of terms may be helpful when reading through this manual: https://www.ueidaq.com/glossary

Additional questions? Please email UEI Support at uei.support@ametek.com or call 508-921-4600.

1.5 Before You Begin

No Hot Swapping!

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

Check Your Firmware

Ensure that the firmware installed on the Cube or Rack CPU matches the UEI software version installed on your PC. The IOM is shipped with pre-installed firmware and a matching software installation. If you upgrade your software installation, you must also update the firmware on your Cube or RACK CPU. See *"Firmware Update Procedures.pdf"* for instructions on checking and updating the firmware. These instructions are located in the following directories:

- On Linux: *PowerDNA_Linux_<x.y.z>/docs*
- On Windows:
  Start » All Programs » UEI » DNx Firmware Update Procedures

1.6 DNx-DIO-480 Features

The DNx-DIO-480 Universal 32 Channel DIO Board supports everything from 3.3 V TTL levels to 55 V industrial signals. The 32 DIO channels are independently configurable and can be set as input or output. The board includes the following features:

- 32 independently reconfigurable channels of industrial digital input and output
  - change of state detection on the inputs
  - digital input data may be streamed to a 32-bit x 2k FIFO
  - digital outputs can be set as current sourcing or current sinking
  - low-level API can configure the DIO-480 to provide data to the FIFO periodically or when there is a change of state
  - Power for the digital outputs can be provided independently for the 4-channel Vcc groups. Each group can have a supply voltage from 3 V to 55 V.
- Two 3.3 V TTL digital inputs and two outputs
- Two counter/timers that may be routed to/from industrial digital I/O or TTL.

1.6.1 Digital I/O

The DNx-DIO-480 includes 32 channels of industrial digital I/O and 4 channels of logic-level I/O (2 input and 2 output).

1.6.1.1 Industrial Bits

The industrial digital I/O subsystem operates across a wide range, from 3.3 V to 55 VDC. Each industrial bit is independently configurable as either input or output. Voltage is supplied in groups of 4 bits (up to 8 different Vcc groups across 32 bits).

Inputs: Each input is sensed with a dedicated 100 kHz A/D converter. High and low thresholds are therefore programmable and state changes can be detected with 5 microsecond resolution. Programmable pull-up and/or pull-down resistors allow inputs to monitor contacts connected to a supply voltage or ground. In the absence of an external supply voltage or programmable pull-up and/or pull-down resistors, the lines are weakly pulled up to an internal 60 V supply (via a 2 MΩ resistor); this ensures that inputs allow the full 0-55 V range, but can be easily overdriven by an external source.

Outputs: Each output may be configured as either current sourcing (connect output to Vcc) or sinking (connect output to Gnd). Outputs are rated for continuous operation at 500 mA with an output voltage drop of less than 600 mV. Each channel is protected with an electronic circuit breaker and a 1.25 A fast-blow fuse.

Industrial digital outputs are equipped with an optional pulse-width modulated (PWM) “soft-start” or “soft-stop” feature. This allows power to be applied or removed gradually, greatly increasing the reliability of devices like incandescent bulbs where thermal shock reduces life expectancy. The ‘soft-start” parameters are selectable on a per-channel basis.

PWM can also be configured to run continuously for low speed, high voltage/ current applications. The board supports pulse-width resolution up to 16-bits and frequency up to 25 kHz.

The DIO-480 allows channel pairs 0-1, 2-3, ..., 30-31 to be configured as H-Bridge pairs. This easily and safely allows support for the four standard states of a DC motor: Drive Right, Drive Left, Brake, Coast.

1.6.1.2 TTL Bits

A total of four 3.3 V isolated logic-level channels are provided: two channels are available as input and two are available as output.

1.6.1.3 Counters

Two 32-bit counters perform up/down counting. Several flexible modes are available including event counting, pulse-width/period measurements, and simplified quadrature decoding. Counter inputs and outputs can be routed to your choice of industrial DIO or TTL DIO pins.

1.6.2 Guardian Diagnostics

UEI’s Guardian Series boards provide real-time monitoring of system functions. The following diagnostic information can be read from the DNx-DIO-480:

- **Digital Input Voltage** - provides the current voltage on each digital input channel.
- **Vcc Supply Voltage** - provides the current supply voltage of each of the eight Vcc groups.
- **Temperature** - returns the surface temperature of the board. The sensor is located in the hottest area of the board.
- **Circuit Breaker Status** - indicates whether the electronic circuit breakers have tripped on any of the 32 DIO channels. The circuit breakers can be enabled or disabled for each channel. The trip point is ~1 A. Additionally, a 1.25 A fast-blow fuse on each output channel provides overcurrent protection.

1.6.3 Isolation & Over-voltage Protection

Isolation of 350 Vrms is offered between the DNx-DIO-480 customer-facing I/O and other I/O boards as well as between the I/O connections and the chassis.

1.6.4 Environmental Conditions

Like all UEI I/O boards, the DNx-DIO-480 offers operation in extreme environments and has been tested to 5 g vibration, 100 g shock, temperatures from -40 °C to +85 °C, and will function at altitudes up to 70,000 feet.

1.6.5 Accessories

The DNx-DIO-480 is supported by UEI’s DNA-CBL-62 general purpose cable and DNA-STP-62 screw terminal panel. See Appendix A for information on the DNA-CBL-62 and DNA-STP-62.

All connections are through a standard 62-pin “D” connector, allowing OEM users to build custom cabling systems with off-the-shelf components.

1.6.6 Software Support

The DNx-DIO-480 includes a software suite supporting Windows, Linux, QNX, VXWorks, RTX, and most other popular real-time operating systems. Windows and Linux users may use the UeiDaq Framework, which provides a simple and complete software interface to all popular programming languages and DAQ applications (e.g., LabVIEW, MATLAB). All software includes example programs that make it easy to cut-and-paste the I/O software into your applications.

1.7 Technical Specifications

The following tables list the technical specifications for the DNx-DIO-480 board. All specifications are for a temperature of 25 °C ± 5 °C unless otherwise stated.

1.7.1 Industrial Digital I/O

Table 1-1 Industrial Digital I/O Specifications

Number of channels / direction	32 bits independently selectable as input or ouput
Digital Input	
Input range	0-55 VDC
Input high / low voltage	Programmable from 0-55 VDC
Input impedance	> 1.1 MΩ
Input open circuit state	98 kΩ pull-up and pull-down resistors are software enabled. Both options can be active simultanuously.
Input protection	±100 VDC
Input A/D sample rate	100 kHz
Digital input sample rate	100 kHz
Digital input FIFO	32 by 2048
Guardian input accuracy	±275 mV
Change of state resolution	10 µs
Input throughput	Up to 25 kHz depending on deployment
Digital Output	
Configurations	Current sink/source, Ground/open, or Vcc/open (Vcc is user provided in banks of 4 bits)
Output drive	500 mA per channel, continuous
Output protection	1.25 amp fast-blow fuse on each output
Output voltage drop	< 600 mV at 500 mA (Incl std 3’ cable)
Output Off impedance	> 1.1 MΩ
Output Off leakage current	< 50 µA (with 55V input)
Output throughput	Up to 25 kHz depending on deployment
PWM output	0 to 100% in 0.0015% increments (16-bit resolution)
PWM cycle rate	up to 25 kHz
Softstart programmability	up to 10 seconds with 150 µs resolution

1.7.2 TTL Digital I/O

Table 1-2 TTL Digital I/O Specifications

Number of channels	4 bits
I/O direction	2 bits input, 2 bits output
Logic level	3.3 V logic

1.7.3 Counter/Timer

Table 1-3 Counter/Timer Specifications

Number of counters	2, routable to/from any I/O pin
Resolution	32 bits
Max frequency	66 MHz for internal input clock 16.5 MHz for external input clock 33 MHz for outputs
Min frequency	no lower limit
Pulse-width/period accuracy	2 internal clock cycles (30 ns) on one or multiple periods
External gate/trigger inputs	1 per counter, programmable polarity

1.7.4 General

Table 1-4 General and Environmental Specifications

Electrical Isolation	350 Vrms Isolation divided into 8 isolated banks of 4 bits
Power Consumption	< 5 W (not including output loads)
Operating Temp. (tested)	-40 °C to +85 °C
Operating Humidity	95%, non-condensing
*Vibration IEC 60068-2-6 IEC 60068-2-64	5 g, 10-500 Hz, sinusoidal 5 g (rms), 10-500 Hz, broadband random
*Shock IEC 60068-2-27	100 g, 3 ms half sine, 18 shocks @ 6 orientations 30 g, 11 ms half sine, 18 shocks @ 6 orientations
MTBF	400,000 hours

*Shock and vibration specifications assume appropriate mounting/installation.

Chapter 2 DIO Functional Description

This section describes the device architecture, hardware, and functionality of the DNx-DIO-480. The following sections are provided in this chapter:

- Digital I/O (Section 2.1)
- Indicators and Connectors (Section 2.2)
- Pinout (Section 2.3)
- Wiring Guidelines (Section 2.4)

2.1 Digital I/O

The DNx-DIO-480 digital I/O board includes 32 industrial I/O channels, four TTL I/O channels (two input and two output), and two counter/timer modules. Each industrial I/O channel is independently configurable. Each TTL I/O channel is also independently configurable. All digital I/O signals are isolated from the FPGA.

2.1.1 Industrial Digital I/O

Figure 2-1 shows a simplified block diagram of the ADC-based digital I/O subsystem. DIO channels may be configured as either input or output.

Inputs are buffered to protect against input loading and are simultaneously sampled at 100 kHz by 14-bit A/D converters (one ADC per DIO channel). The control logic compares the ADC voltage to user-defined high and low thresholds and returns the digital state. Inputs may also be debounced with programmable delays.

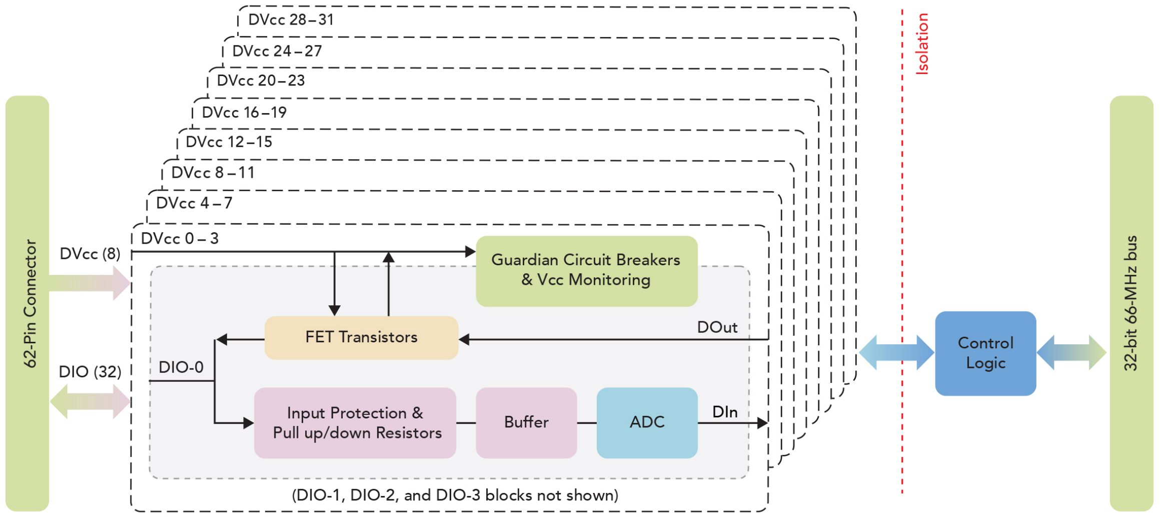


Figure 2-1 Block Diagram of DNx-DIO-480 Industrial Digital I/O

Outputs are switched by a FET-based circuit (**Figure 2-2**) and require an external DC power supply. Up to eight different +DVccs may be supplied to the DNx-DIO-480 board. Users should ensure that each +DVcc can supply enough current for all four channels it powers, up to 500 mA max/channel.

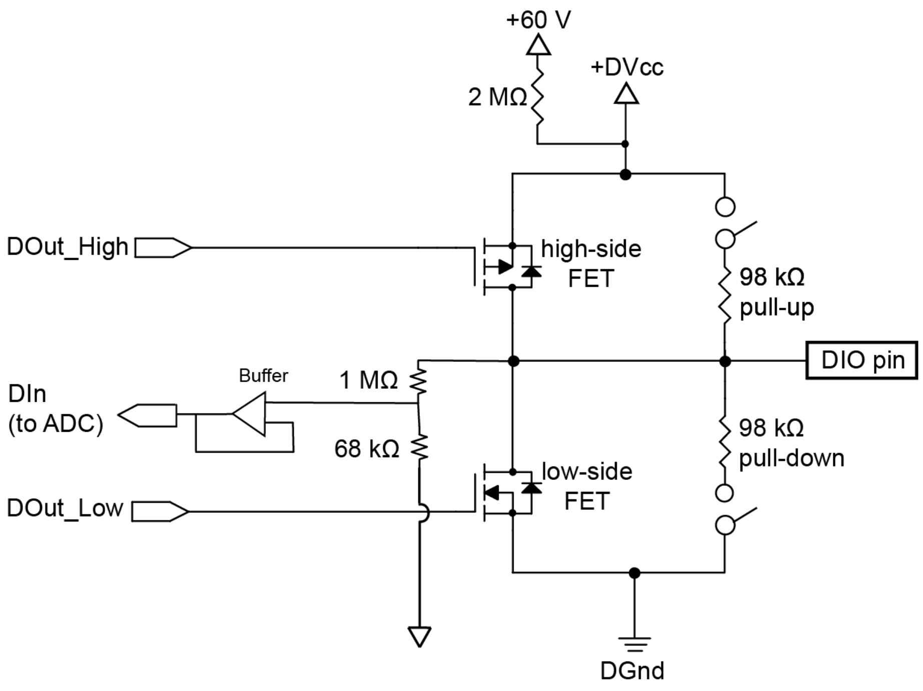


Figure 2-2 Simplified Circuit Diagram of an Industrial DIO Channel

As illustrated in **Figure 2-2**, each output is set to LOW, HIGH, or OFF by a high-side/low-side pair of FETs. When the FPGA writes a 1 on the DOut_HIGH line, the high-side FET turns on and connects the DIO pin to +DVcc (current sourcing). When a 1 is written to the DOut_LOW line, the low-side FET connects the DIO pin to DGnd (current sinking). The control logic prevents both FETs from being on simultaneously. When both high- and low-side FETs are disabled, the pin can be used as a dedicated input.

Each pin’s open-circuit state is software programmable to DVcc, Gnd, or DVcc/2. This is achieved by connecting the pin to an internal 98 kΩ pull-up resistor, 98 kΩ pull-down resistor, or both resistors, respectively.

NOTE: The industrial digital output channels do NOT include built-in anti-kickback diodes. If the channel is used to source or sink an inductive load, we recommend connecting an external diode to protect the FETs against induced voltage spikes (see Section 2.4.1 for wiring information).

If +DVcc is disconnected, the positive rail is automatically pulled up to an internal +60 V supply by a 2 MΩ resistor. The internal supply prevents accidental floating inputs and allows digital inputs to work properly without a user-supplied +DVcc. A user-supplied +DVcc is only required for digital outputs.

When pulled up to the +60 V supply, an unused DIO pin will have some voltage under 60 V (this varies with the number of DO pins driving HIGH). The large 2 MΩ pull-up resistance protects user equipment from this voltage. To set the unused pin to zero, you can add an external 100 kΩ pull-down resistor or enable the internal 98 kΩ pull-down resistor in software. Alternatively, connect Vcc of the 4-bit group to a known supply voltage that will be above the expected maximum voltage on all the DIn pins of the 4-bit group.

2.1.1.1 Pulse Width Modulation

The DNx-DIO-480 offers built-in pulse width modulation (PWM) on industrial digital outputs. PWM mode, frequency, duty cycle, and push/pull mode are per channel configurable.

The following PWM modes are available:

- **Continuous PWM** - The duty cycle is constant over the entire period of operation. This is normal PWM output. A typical application for this feature is a dimmer for an incandescent indicator light in which the average voltage applied to a bulb is increased or decreased by varying the PWM duty cycle.
- **Soft Start** - As shown in **Figure 2-3**, a soft start increases the PWM duty cycle gradually from 0% up to the configured steady-state value. This feature is useful in preventing premature burnout of devices (such as incandescent bulbs) caused by too rapid heating on startup.
- **Soft Stop** - Soft stop is the opposite of soft start. The duty cycle decreases gradually down to 0% when the output transitions from HIGH to LOW. The typical application for soft stop mode is a soft start operation that is implemented with inverted logic.
- **Soft Both** - Configures PWM for both soft start and soft stop.
- **Gated** - Normal PWM is enabled when the channel is written HIGH.

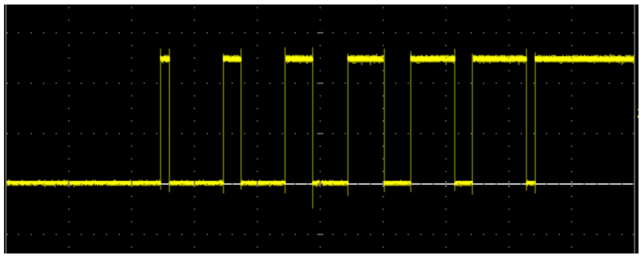
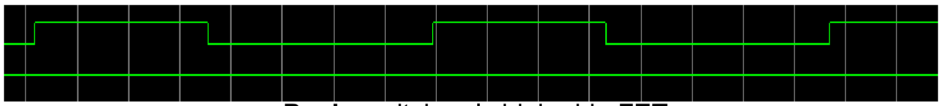
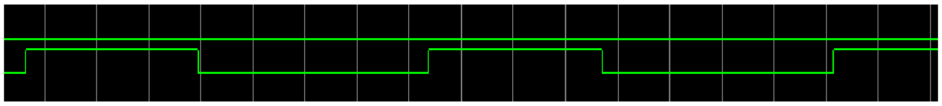


Figure 2-3 Typical PWM Soft Start cycle

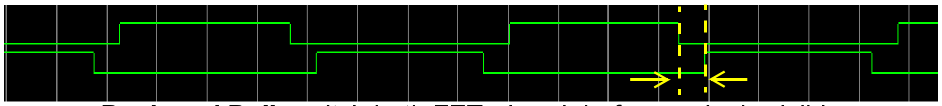
A PWM output can be configured to switch one or both FETs in the channel. A break-before-make interval prevents both FETs from being on at the same time, as shown in **Figure 2-4**.



Push: switch only high-side FET



Pull: switch only low-side FET



Push and Pull: switch both FETs; break-before-make is visible

Figure 2-4 PWM Push/Pull output modes

It is also possible to generate pulse trains using the counters described in Section 2.1.3. However, the built-in PWM system is easier to use and therefore recommended for industrial digital outputs.

2.1.1.2 H-Bridge Support

The DIO-480 allows channel pairs 0-1, 2-3, ..., 30-31 to be configured as H-Bridge pairs. This easily and safely allows support for the four standard states of a DC motor: Drive Right, Drive Left, Brake, Coast. See the Programming chapters for more information on H-Bridge configuration.

2.1.1.3 Digital Output Diagnostics

Because DOut and DIn share the same pin, the board can read back DOut voltage through the DIn ADC. In addition to the software enabled circuit breakers on each digital output channel, the board provides overcurrent protection using a 1.25 A fast-blow fuse on each output channel.

2.1.2 TTL Digital IO

The TTL bits use 3.3 V logic levels (an input above 2.3 V is HIGH, while a voltage below 1 V is LOW). The DNx-DIO-480 is capable of single read/write into the registers as well as continuous clock reads and writes. PWM signals can be generated on TTL outputs via the counter subsystem described in Section 2.1.3.

2.1.3 Counters

Industrial and TTL DIO pins may be routed to two 32-bit counters in order to perform a number of customizable operations including:

- **Timer:** counts off a user-defined time interval
- **Event Counter:** counts the number of rising or falling edges on a signal
- **Bin Counter:** counts the number of pulses in the specified time interval
- **Pulse-Width/Period:** measures the width of the positive and/or negative parts of the input signal
- **Timed Pulse Period Measurement:** measures average frequency of incoming pulses over a user-defined time interval

- **Quadrature Decoder:** measures relative position from a quadrature encoder sensor
- **PWM Generator:** outputs a pulse-width-modulated waveform and update its period and duty cycle on the fly.

As shown in **Figure 2-5**, each counter has three lines:

- **Input clock (CLKIN):** takes in the signal to be measured
- **Output clock (CLKOUT):** drives one or more digital output pins according to the counter's mode of operation
- **Gate/Trigger input (GATE):** takes in a gating signal, start/stop/restart trigger, or the quadrature encoder direction.

Both input lines are connected to de-bouncers to eliminate unwanted spikes in the signals. The counter counts up to 2^32 and can be clocked by either **CLKIN**, a 66 MHz internal base clock, or a divided version of either clock.

NOTE: If the counter is routed to industrial digital inputs, the measurement resolution is limited by the 100 kHz DIn ADC clock rate (e.g., pulse width will be returned in 2.5 µs increments). TTL-level inputs do not use the ADC and can therefore be measured down to 15 ns.

The counter's behavior is defined according to the values of the registers shown in **Figure 2-5** and described in **Table 2-1**. Refer to Chapter 4 and Chapter 5 for information about configuring the counting modes.

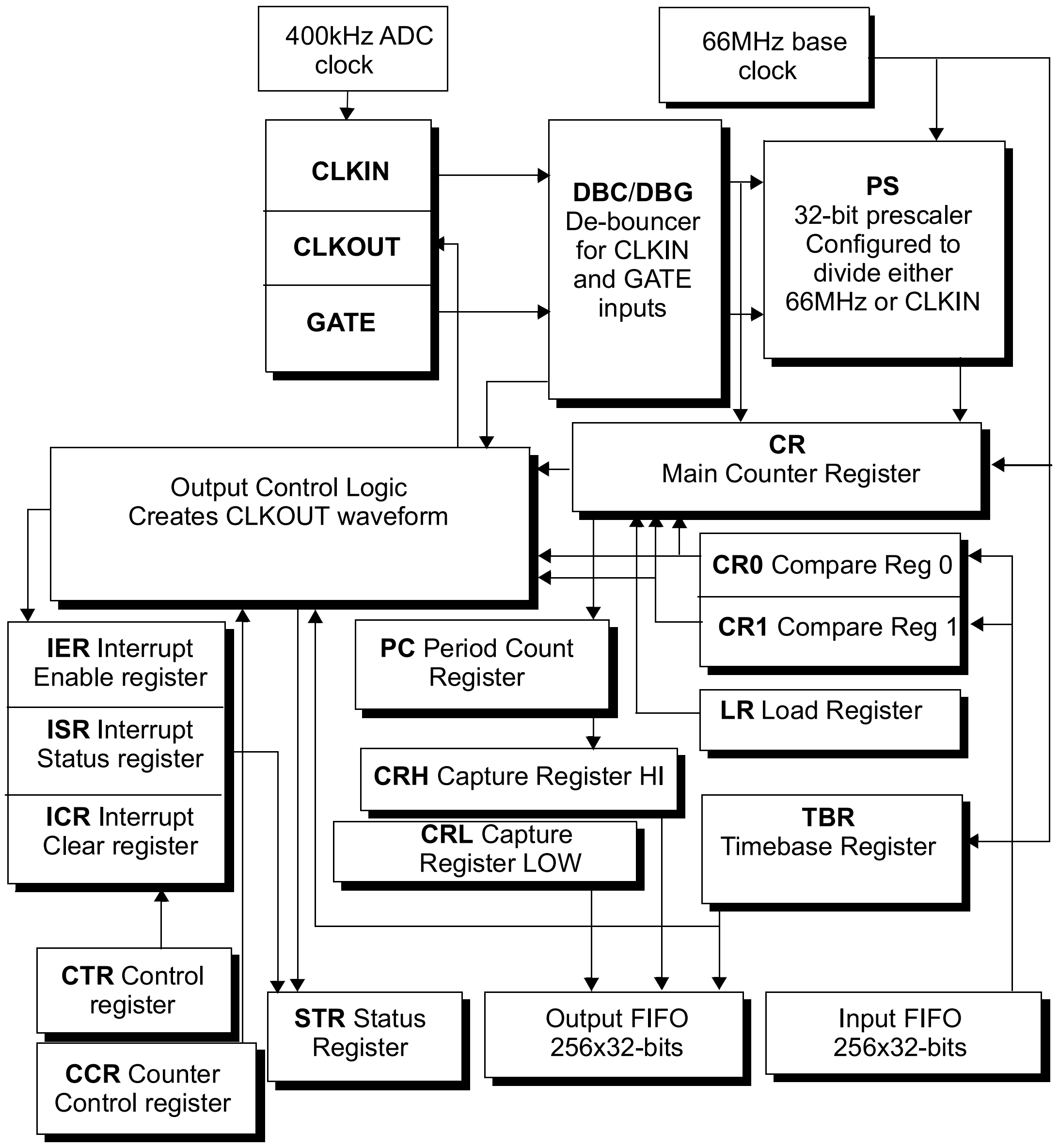


Figure 2-5 Internal Structure of DNx-DIO-480 Counter

Table 2-1 DNx-DIO-480 Counter Registers

Reg	Name	Description
CCR	Counter Control Register	defines the operation mode of the counter and prescaler
CR	Main Counter Register	stores the count; counts upward in all modes except for quadrature decoder mode which allows both up and down counting
CR0	Compare Register 0	defines how long CLKOUT stays low
CR1	Compare Register 1	defines how long CLKOUT stays high
CRH	Capture Register HIGH	used when the counter measures parameters of the CLKIN signal
CRL	Capture Register LOW	used when the counter measures parameters of the CLKIN signal
CTR	Control Register	enables/disables the counter, enables/disables inversion mode for I/O pins and buffered FIFO operation
ICR	Interrupt Mask Register	clears interrupt condition(s) after a CPU processes them
DBC	CLKIN De-bouncing Register	defines number of 66 MHz clock cycles for which the Input Clock signal must be stable
DBG	GATE De-bouncing Register	defines number of 66 MHz clock cycles for which the Gate signal must be stable
IER	Interrupt Enable Register	enables/disables interrupt generation; 16 interrupt conditions are available
ISR	Interrupt Status Register	reports status of the enabled interrupts
LR	Load Register	stores the initial value from which the counter starts counting
PC	Period Count Register	used when measuring a signal that is too fast to read every period; data from **CR** is supplied only when measured data has accumulated over N periods
STR	Status Register	reports current status of the counter operation
TBR	Timebase Register	defines the measurement time interval in certain modes

2.2 Indicators and Connectors

Figure 2-6 shows the locations of the LEDs and connectors on the DNx-DIO-480. The LED indicators are described below in Table 2-2.

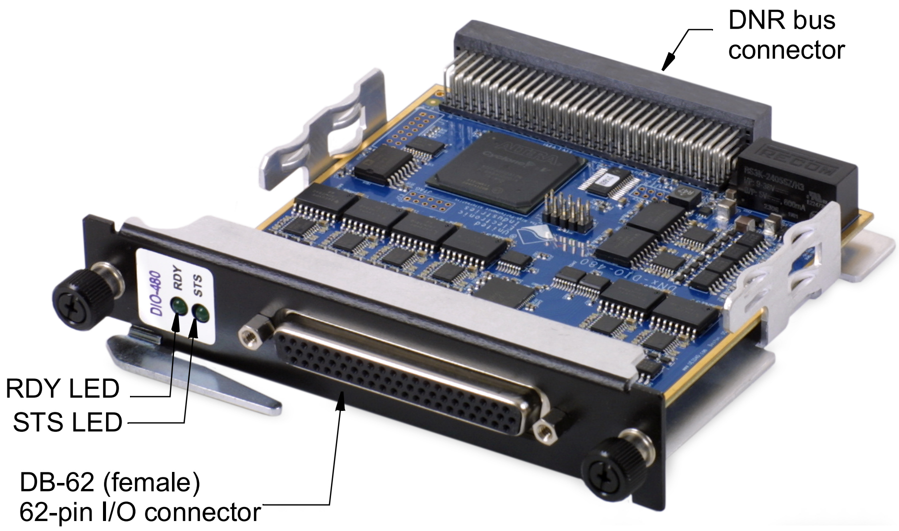


Figure 2-6 Photo of DNR-DIO-480 Board

Table 2-2 LED Indicators

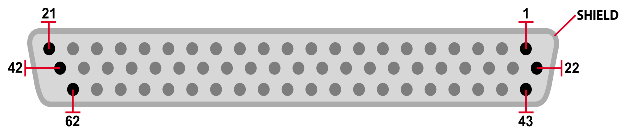
LED Name	Description
RDY	**READY:** board is powered up and operational
STS	**STATUS:** **OFF**: Configuration mode (e.g., configuring channels, running in Point-by-Point mode) **ON**: Operation mode (e.g., running in DMap or VMap mode)

2.3 Pinout

Figure 2-7 illustrates the pin configuration for the DNx-DIO-480 board. Connections are made through a standard DB-62 female connector.

Signals are isolated in three groups:

- **DIO** is referenced to Gnd. Use Gnd as the return for DIO-n and Vcc n-m.
- **Gnd** - returns for the industrial DIO. UEI recommends to utilize the maximum number of grounds when designing your cabling system. Each pin on the DIO-480 can carry a maximum of 2 A of continuous current.
- **Vcc** n-m is the user supplied Vcc for industrial DO channels n-m. The digital outputs are divided into 8 groups of 4. If you desire to provide a Vcc for the digital output to switch on/off, you have the option of using more than one drive voltage.
  Note: For continuous high-current applications, it is recommended to use DIO 0:3 and DIO 16:19 because they offer two Vcc pins per 4-bit group, thereby reducing stress on the cabling.
- **TTL DIO** uses a separate isolated GND from the DIO 0:31 signals. This is done to minimize crosstalk between the industrial and TTL DIO. TTL-VDD is a 3.3 V output capable of sourcing up to 60 mA.



Pin	Signal	Pin	Signal	Pin	Signal
1	DIO 0	22	Gnd	43	Gnd
2	DIO 1	23	Vcc 0-3	44	DIO 2
3	DIO 3	24	Gnd	45	VCC 0-3
4	DIO 4	25	Vcc 4-7	46	DIO 5
5	DIO 6	26	Gnd	47	DIO 7
6	DIO 8	27	Vcc 8-11	48	DIO 9
7	DIO 10	28	Gnd	49	DIO 11
8	DIO 12	29	Vcc 12-15	50	DIO 13
9	DIO 14	30	Gnd	51	DIO 15
10	DIO 16	31	Gnd	52	Gnd
11	DIO 17	32	Vcc 16-19	53	DIO 18
12	DIO 19	33	Gnd	54	Vcc 16-19
13	DIO 20	34	Vcc 20-23	55	DIO 21
14	DIO 22	35	Gnd	56	DIO 23
15	DIO 24	36	Vcc 24-27	57	DIO 25
16	DIO 26	37	Gnd	58	DIO 27
17	DIO 28	38	Vcc 28-31	59	DIO 29
18	DIO 30	39	Gnd	60	DIO 31
19	reserved	40	Gnd	61	TTL DOUT 0
20	TTL DOUT 1	41	TTL (3.3VDD)	62	TTL DIN 0
21	TTL DIN 1	42	TTL Gnd		

Figure 2-7 Pinout Diagram for DNx-DIO-480

Table 2-3 shows the recommended return paths for industrial and TTL DIO lines.

Table 2-3 Recommended Return Paths

Signal	Recommended Return
TTL DIN 0:1	Pin 42
TTL DOUT 0:1	Pin 42
DIO 0-3	Pins 43 and 24
DIO 4-7	Pin 26
DIO 8-11	Pin 28
DIO 12-15	Pin 30
DIO 16-19	Pins 31 and 52
DIO 20-23	Pin 35
DIO 24-27	Pin 37
DIO 28-31	Pin 39

No Hot Swapping!

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

2.4 Wiring Guidelines

The following wiring schemes are recommended when connecting external devices to the DNx-DIO-480.

2.4.1 Industrial Digital Output Wiring

When using the industrial digital output subsystem, ensure that DVcc is connected to the user's power supply (0-55 VDC). A disconnected DVcc will not damage the DNx-DIO-480 but may cause unexpected digital input readings as the outputs switch ON/OFF.

A load may be wired to the output in any of the following configurations:

- **Push Mode:** DNx-DIO-480 acts as a switch between DVcc and the output, sourcing current to the load when the switch is on. An example circuit is shown in **Figure 2-8**a.
- **Pull Mode:** DNx-DIO-480 acts as a switch between the output pin and DGnd, sinking current from the load when the switch is on. An example circuit is shown in **Figure 2-8**b.
- **Push-Pull Mode:** DNx-DIO-480 connects the output to either DVcc or DGnd, never both at the same time. An example dual-channel circuit is shown in **Figure 2-8**c. Current flows through the solenoid when one channel is set HIGH and the other channel is set LOW. The current is easily reversed by inverting the outputs.

Note that the diagrams in **Figure 2-8** include an optional external anti-kickback/ flyback diode. UEI recommends adding the diode when driving inductive loads such as relays or solenoids. Without the diode, a large voltage spike can occur across the inductive load when its supply current is suddenly shut off, potentially damaging the FET switch inside the DNx-DIO-480. The anti-kickback diode provides an alternate path for the current and clamps the voltage spike to a safe value.

The diode in Push Mode or Pull Mode can be a general purpose diode rated to handle the steady-state current through the inductor and the desired switching speed. Connect the Push Mode or Pull Mode diode parallel to the load.

In Push-Pull Mode, we suggest using a bidirectional transient-voltage-suppression (TVS) diode such as the P6KE68CA. Connect the TVS diode from the FET line to Gnd.

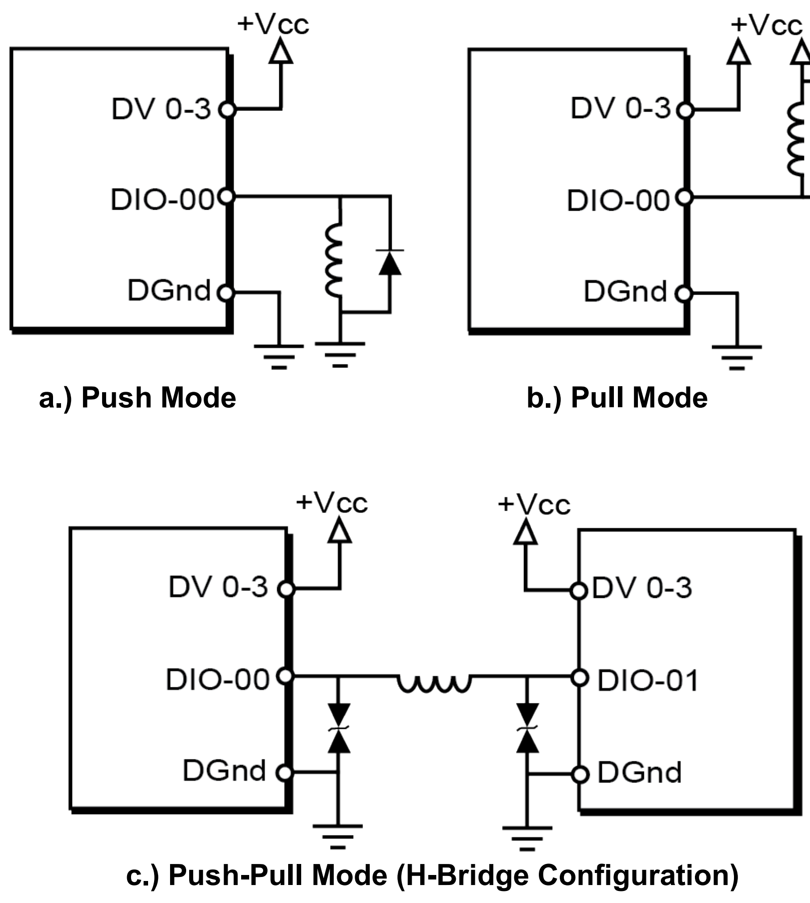


Figure 2-8 Industrial Digital Output Wiring

Chapter 3 PowerDNA Explorer

This chapter provides the following information about exploring the DNx-DIO-480 with the PowerDNA Explorer application.

- Introduction (Section 3.1)
- Industrial Digital Input (Section 3.2)
- Industrial Digital Output (Section 3.3)
- Configure Channel Termination (Section 3.4)
- Counter/Timer (Section 3.5)
- TTL-Level DIO (Section 3.6)

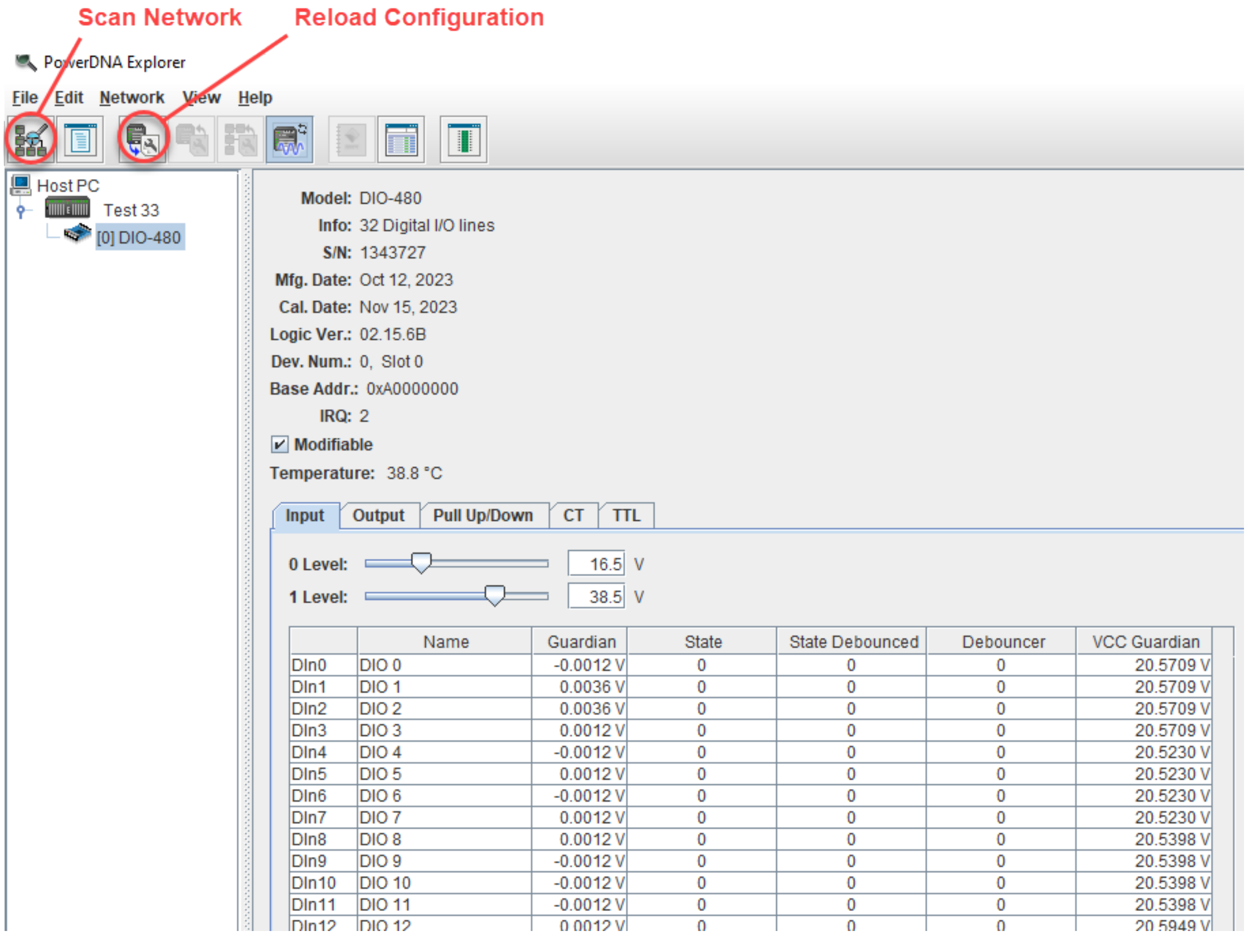
3.1 Introduction

PowerDNA Explorer is a GUI-based application for communicating with your RACK or Cube system. You can use it to start exploring a system and individual boards in the system. PowerDNA Explorer can be launched from the Windows startup menu:

Start » All Programs » UEI » PowerDNA Explorer

The DNx-DIO-480 is supported in PowerDNA version 5.2.0.8+.

When using PowerDNA Explorer to configure DNx-DIO-480 boards, resetting the IOM or changing the DNx-DIO-480 configuration outside of PowerDNA Explorer (e.g., via C code or Labview) is not recommended. PowerDNA Explorer will not display changed parameters until **Scan Network** or **Reload Configuration** is clicked again (see **Figure 3-1** for button locations).



Model: DIO-480
Info: 32 Digital I/O lines
S/N: 1343727
Mfg. Date: Oct 12, 2023
Cal. Date: Nov 15, 2023
Logic Ver.: 02.15.6B
Dev. Num.: 0, Slot 0
Base Addr.: 0xA0000000
IRQ: 2
[✓] **Modifiable**
Temperature: 38.8 °C

0 Level: 16.5 V
1 Level: 38.5 V

	Name	Guardian	State	State Debounced	Debouncer	VCC Guardian
DIn0	DIO 0	-0.0012 V	0	0	0	20.5709 V
DIn1	DIO 1	0.0036 V	0	0	0	20.5709 V
DIn2	DIO 2	0.0036 V	0	0	0	20.5709 V
DIn3	DIO 3	0.0012 V	0	0	0	20.5709 V
DIn4	DIO 4	-0.0012 V	0	0	0	20.5230 V
DIn5	DIO 5	0.0012 V	0	0	0	20.5230 V
DIn6	DIO 6	-0.0012 V	0	0	0	20.5230 V
DIn7	DIO 7	0.0012 V	0	0	0	20.5230 V
DIn8	DIO 8	0.0012 V	0	0	0	20.5398 V
DIn9	DIO 9	-0.0012 V	0	0	0	20.5398 V
DIn10	DIO 10	-0.0012 V	0	0	0	20.5398 V
DIn11	DIO 11	-0.0012 V	0	0	0	20.5398 V
DIn12	DIO 12	0.0012 V	0	0	0	20.5949 V

Figure 3-1 PowerDNA Explorer for DNx-DIO-480

When the DNx-DIO-480 is selected in the left-hand panel, the right-hand panel contains a tab for each subsystem as well as a tab for configuring the pull up/ pull down resistors.

- **Input**: read digital inputs and diagnostic ADCs
- **Output**: configure industrial digital outputs, including PWM
- **Pull Up/Down**: select termination resistors for industrial digital outputs
- **CT**: configure counter/timer sources and counting modes
- **TTL**: configure TTL digital outputs and read input port

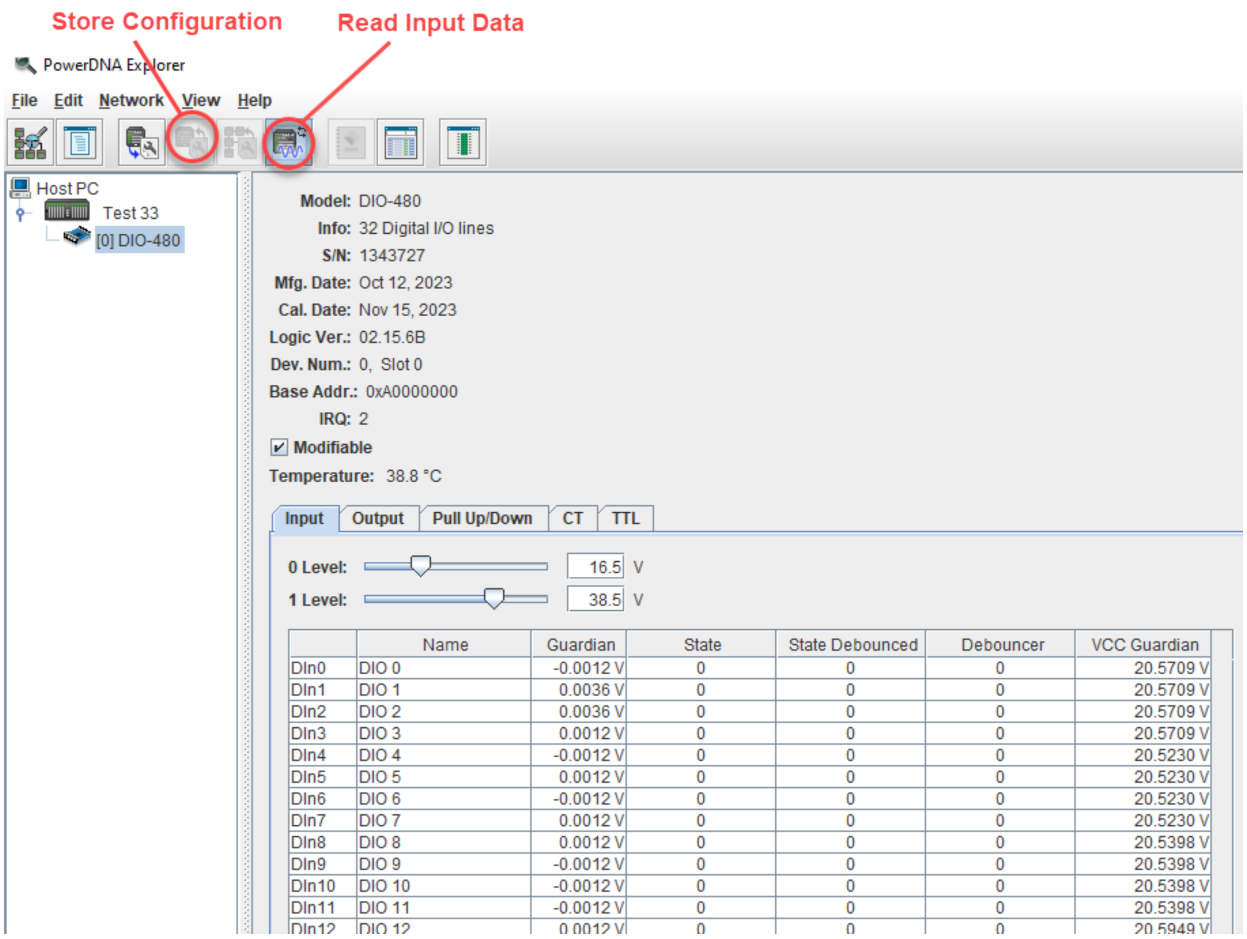
3.2 Industrial Digital Input

To explore the industrial digital input subsystem, select the Input tab (**Figure 3-2**).

Click **Read Input Data** to start data acquisition.

The Input tab contains the following controls and columns:

- **0 Level**: slider and numeric text field for setting the logic level low threshold (between 0 to 55 V). The logic level changes from 1 to 0 when the input voltage transitions below the 0 Level. Click **Store Configuration** for the changes to take effect.
- **1 Level**: slider and numeric text field for setting the logic level high threshold (between 0 to 55 V). The logic level changes from 0 to 1 when the input voltage transitions above the 1 Level. Click **Store Configuration** for the changes to take effect.
- **DInX**: read-only display of the channel number.
- **Name**: a name or note that you wish to give to the channel.
- **Guardian**: displays the voltage data from the channel's ADC.
- **State**: displays the current state of the channel. This state is determined by comparing the ADC voltage to the configured 0 Level and 1 Level.
- **State Debounced**: displays the debounced state of the channel. This logic level must have held steady over the number of samples defined in the "Debouncer" column. Click **Store Configuration** for the changes to take effect.
- **Debouncer**: numeric text field to set the debouncing interval for the channel. This is the number of ADC samples required for a debounced state change (max 65535).
- **Vcc Guardian**: displays the current supply voltage for each Vcc group.



Model: DIO-480
Info: 32 Digital I/O lines
S/N: 1343727
Mfg. Date: Oct 12, 2023
Cal. Date: Nov 15, 2023
Logic Ver.: 02.15.6B
Dev. Num.: 0, Slot 0
Base Addr.: 0xA0000000
IRQ: 2
[✓] Modifiable
Temperature: 38.8 °C

0 Level: 16.5 V
1 Level: 38.5 V

	Name	Guardian	State	State Debounced	Debouncer	VCC Guardian
DIn0	DIO 0	-0.0012 V	0	0	0	20.5709 V
DIn1	DIO 1	0.0036 V	0	0	0	20.5709 V
DIn2	DIO 2	0.0036 V	0	0	0	20.5709 V
DIn3	DIO 3	0.0012 V	0	0	0	20.5709 V
DIn4	DIO 4	-0.0012 V	0	0	0	20.5230 V
DIn5	DIO 5	0.0012 V	0	0	0	20.5230 V
DIn6	DIO 6	-0.0012 V	0	0	0	20.5230 V
DIn7	DIO 7	0.0012 V	0	0	0	20.5230 V
DIn8	DIO 8	0.0012 V	0	0	0	20.5398 V
DIn9	DIO 9	-0.0012 V	0	0	0	20.5398 V
DIn10	DIO 10	-0.0012 V	0	0	0	20.5398 V
DIn11	DIO 11	-0.0012 V	0	0	0	20.5398 V
DIn12	DIO 12	0.0012 V	0	0	0	20.5949 V

Figure 3-2 PowerDNA Explorer Input Tab

3.3 Industrial Digital Output

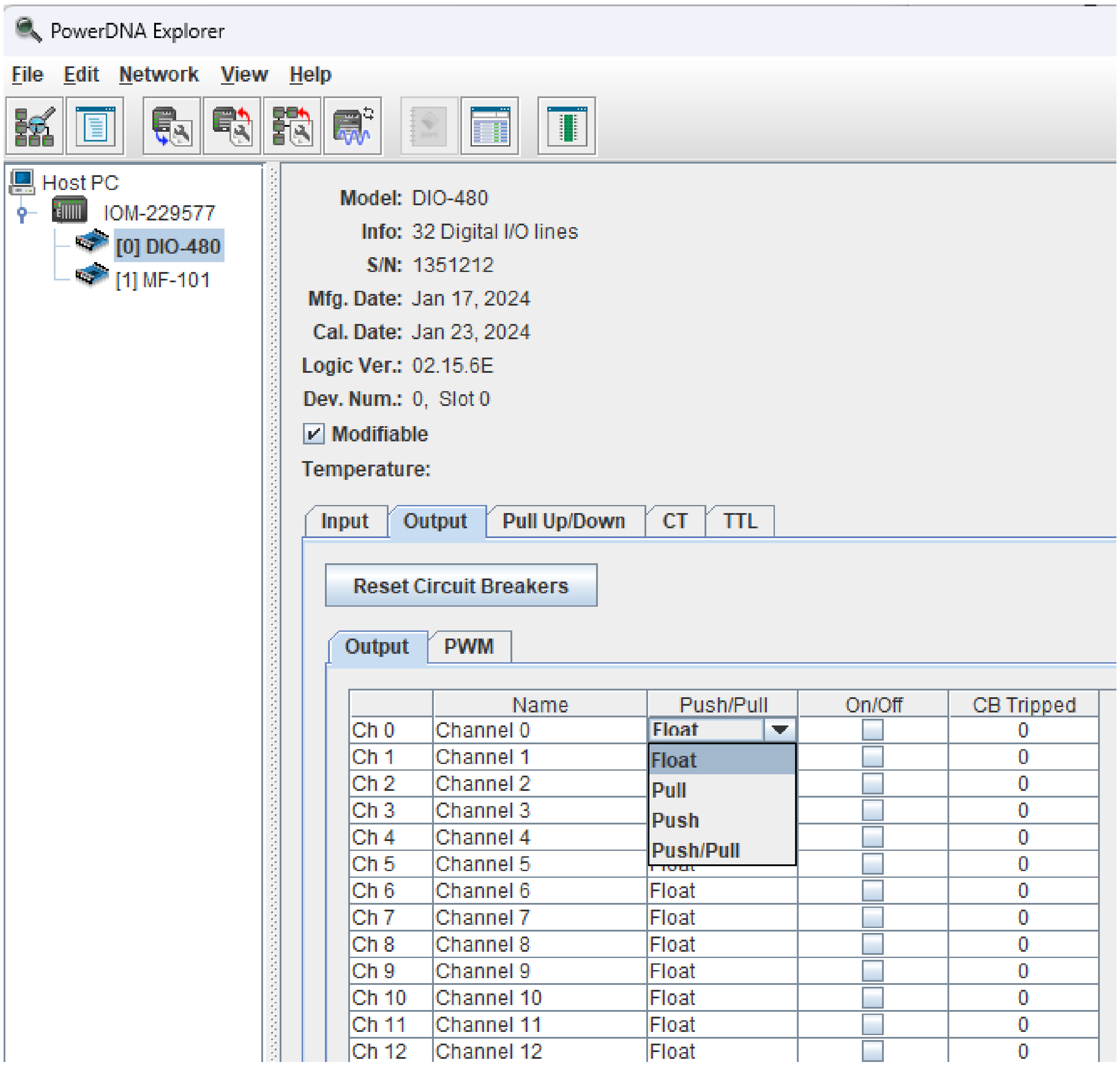
To explore the industrial digital output subsystem, select the Output tab. The Output tab contains the Output and PWM subtabs and the Reset Circuit Breakers button.

3.3.1 Write to Digital Output

The Output subtab (**Figure 3-3**) displays the following columns:

- **Ch X**: read-only display of the channel number.
- **Name**: a name or note that you wish to give to the channel.
- **Push/Pull**: sets the output mode for the channel.
  - **Float**: Channel will always be floating. Use this mode if the channel will be used as an input.
  - **Pull**: Channel will be floating when set to 1, and low when set to 0.
  - **Push**: Channel will be high when set to 1, and floating when set to 0.
  - **Push/Pull**: Channel will be high when set to 1, and low when set to 0.
- **On/Off**: Sets the output state to 1 when selected, 0 when not selected.
- **CB Tripped**: indicates if the circuit breaker has been tripped for the channel. Select the “Reset Circuit Breakers” button to reset all circuit breakers.

.



Model: DIO-480
Info: 32 Digital I/O lines
S/N: 1351212
Mfg. Date: Jan 17, 2024
Cal. Date: Jan 23, 2024
Logic Ver.: 02.15.6E
Dev. Num.: 0, Slot 0
☑ Modifiable
Temperature:

	Name	Push/Pull	On/Off	CB Tripped
Ch 0	Channel 0	Float	☐	0
Ch 1	Channel 1		☐	0
Ch 2	Channel 2		☐	0
Ch 3	Channel 3		☐	0
Ch 4	Channel 4		☐	0
Ch 5	Channel 5		☐	0
Ch 6	Channel 6	Float	☐	0
Ch 7	Channel 7	Float	☐	0
Ch 8	Channel 8	Float	☐	0
Ch 9	Channel 9	Float	☐	0
Ch 10	Channel 10	Float	☐	0
Ch 11	Channel 11	Float	☐	0
Ch 12	Channel 12	Float	☐	0

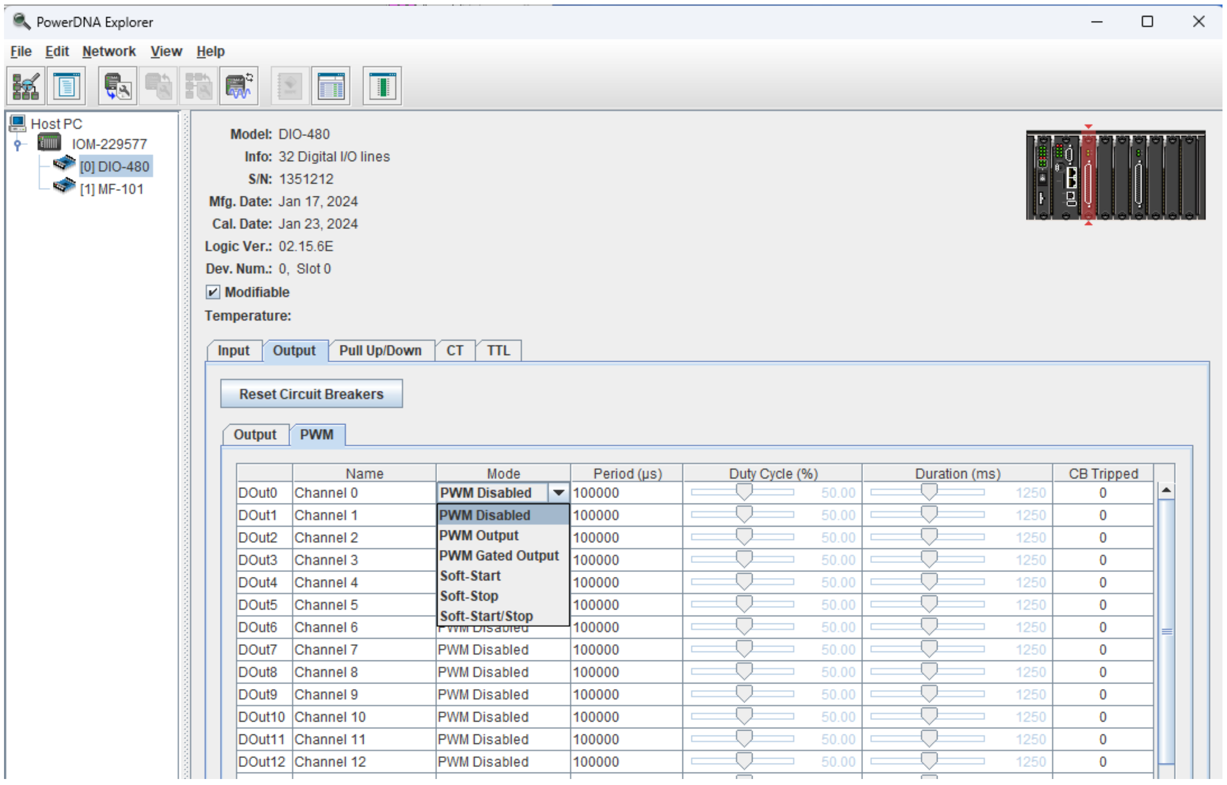
Figure 3-3 PowerDNA Explorer Output Tab, Output Subtab

3.3.2 Configure PWM

The PWM subtab on the Output tab (**Figure 3-4**) is used to configure the PWM mode for each channel. The subtab displays the following columns:

- **DOutX**: read-only display of the channel number.
- **Name**: a name or note that you wish to give to the channel.
- **Mode**: one of the following PWM modes:
  - **PWM Disabled**: disables PWM on the output channel
  - **PWM Output**: enables PWM on the output channel
  - **PWM Gated Output**: enables gated PWM output on the channel. PWM starts when the output is 1 and stops when it is 0.

  - **Soft-Start**: When Output is switched from LOW to HIGH, the duty cycle gradually increases from 0% to the percentage specified in the Duty Cycle column over the specified Duration.
  - **Soft-Stop**: When Output is switched from HIGH to LOW, the duty cycle gradually decreases from the percentage specified in the Duty Cycle column to 0% over the specified Duration.
  - **Soft-Start/Stop**: Selecting this will cause PWM to both soft start and soft stop.
- **Period**: the period of the pulse-width modulated output in microseconds. Type in a number, press the **Enter** key, and click the **Store Configuration** button.
- **Duty Cycle (%)**: Defines the duty cycle for "PWM Output" mode and the soft start and soft stop modes.
- **Duration (ms)**: Defines the duration of the full "Soft-Start" or "Soft-Stop" cycle in milliseconds. This duration should be set longer than the PWM period.
- **CB Tripped**: indicates if the circuit breaker has been tripped for the channel. Select the "Reset Circuit Breakers" button to reset all circuit breakers.



Model: DIO-480
Info: 32 Digital I/O lines
S/N: 1351212
Mfg. Date: Jan 17, 2024
Cal. Date: Jan 23, 2024
Logic Ver.: 02.15.6E
Dev. Num.: 0, Slot 0
✓ Modifiable
Temperature:

	Name	Mode	Period (µs)	Duty Cycle (%)	Duration (ms)	CB Tripped
DOut0	Channel 0	PWM Disabled	100000	50.00	1250	0
DOut1	Channel 1		100000	50.00	1250	0
DOut2	Channel 2		100000	50.00	1250	0
DOut3	Channel 3		100000	50.00	1250	0
DOut4	Channel 4		100000	50.00	1250	0
DOut5	Channel 5		100000	50.00	1250	0
DOut6	Channel 6	PWM Disabled	100000	50.00	1250	0
DOut7	Channel 7	PWM Disabled	100000	50.00	1250	0
DOut8	Channel 8	PWM Disabled	100000	50.00	1250	0
DOut9	Channel 9	PWM Disabled	100000	50.00	1250	0
DOut10	Channel 10	PWM Disabled	100000	50.00	1250	0
DOut11	Channel 11	PWM Disabled	100000	50.00	1250	0
DOut12	Channel 12	PWM Disabled	100000	50.00	1250	0

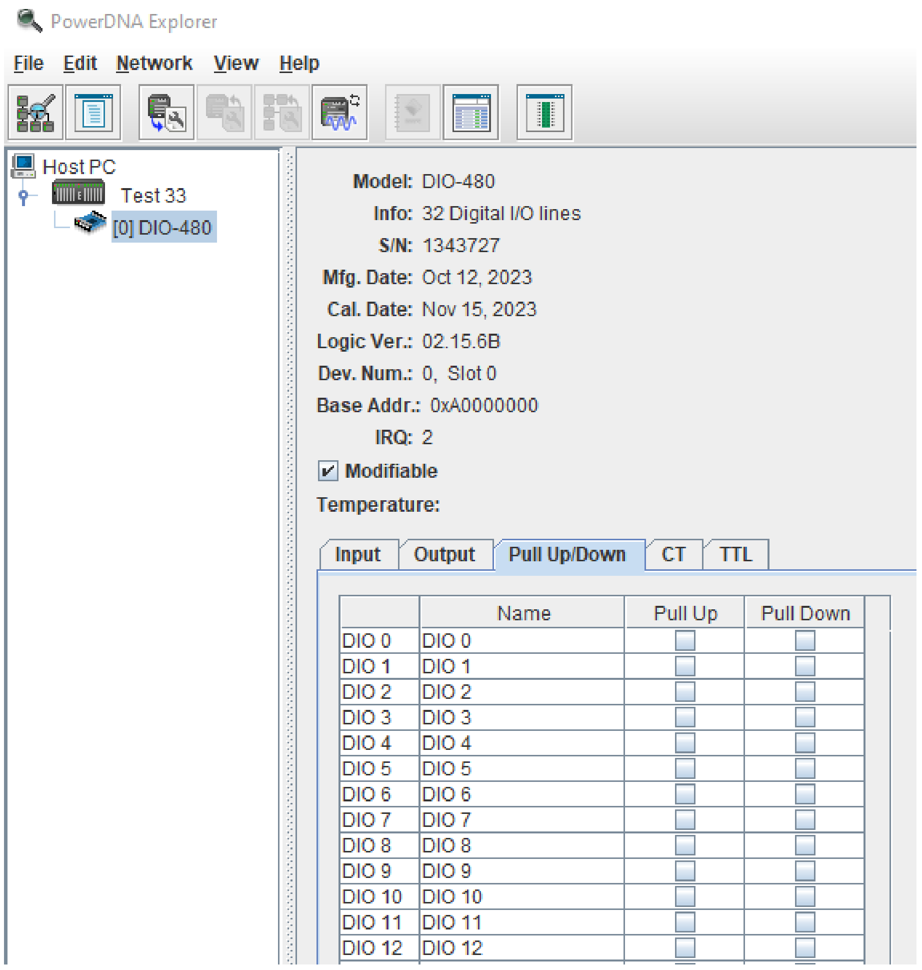
Figure 3-4 PowerDNA Explorer Output Tab, PWM Subtab

3.4 Configure Channel Termination

The Pull Up/Down tab (**Figure 3-5**) allows configuration of each channel's pull-up and pull-down resistors.

- **Ch X**: read-only display of the channel number.
- **Name**: a name or note that you wish to give to the channel.
- **Pull Up**: enables the channel's pull-up resistor. The pull-up resistor pulls up to the channel's Vcc line.
- **Pull Down**: enables the channel's pull-down resistor. The pull-down resistor pulls down to GND.

For each channel, both the pull-up and pull-down resistors can be enabled simultaneously.



Model: DIO-480
Info: 32 Digital I/O lines
S/N: 1343727
Mfg. Date: Oct 12, 2023
Cal. Date: Nov 15, 2023
Logic Ver.: 02.15.6B
Dev. Num.: 0, Slot 0
Base Addr.: 0xA0000000
IRQ: 2
☑ Modifiable
Temperature:

	Name	Pull Up	Pull Down
DIO 0	DIO 0	☐	☐
DIO 1	DIO 1	☐	☐
DIO 2	DIO 2	☐	☐
DIO 3	DIO 3	☐	☐
DIO 4	DIO 4	☐	☐
DIO 5	DIO 5	☐	☐
DIO 6	DIO 6	☐	☐
DIO 7	DIO 7	☐	☐
DIO 8	DIO 8	☐	☐
DIO 9	DIO 9	☐	☐
DIO 10	DIO 10	☐	☐
DIO 11	DIO 11	☐	☐
DIO 12	DIO 12	☐	☐

Figure 3-5 PowerDNA Explorer Pull Up/Down Tab

3.5 Counter/ Timer

To explore the counter/timer modules, open the CT tab.

3.5.1 Configure Count Mode and Sources

The DNx-DIO-480 includes two independent counter/timer modules. Counter/ timer configuration and operation depends on the selected mode. PowerDNA Explorer supports the following modes:

- **Quadrature**: counts pulses on the external input. The count increases or decreases depending on the Gate signal. (Section 3.5.2)
- **Bin Counter:** counts pulses on the external input over a 1 second interval. (Section 3.5.3)
- **PWM Output:** generates a square wave on the output. (Section 3.5.4)
- **Frequency:** measures the frequency of the external Input over a user-configured time interval. (Section 3.5.5)

You can route the counter's Gate, Input, and Output lines to any FET-based DIO or TTL DIO source listed in the drop-down menu. Click **Store Configuration** to program the source settings in hardware.

NOTE: When using FET-based sources as the Input or Gate, always configure digital input levels on the Input tab (Section 3.2) and click **Start Reading Input Data** to enable the DI ADC.

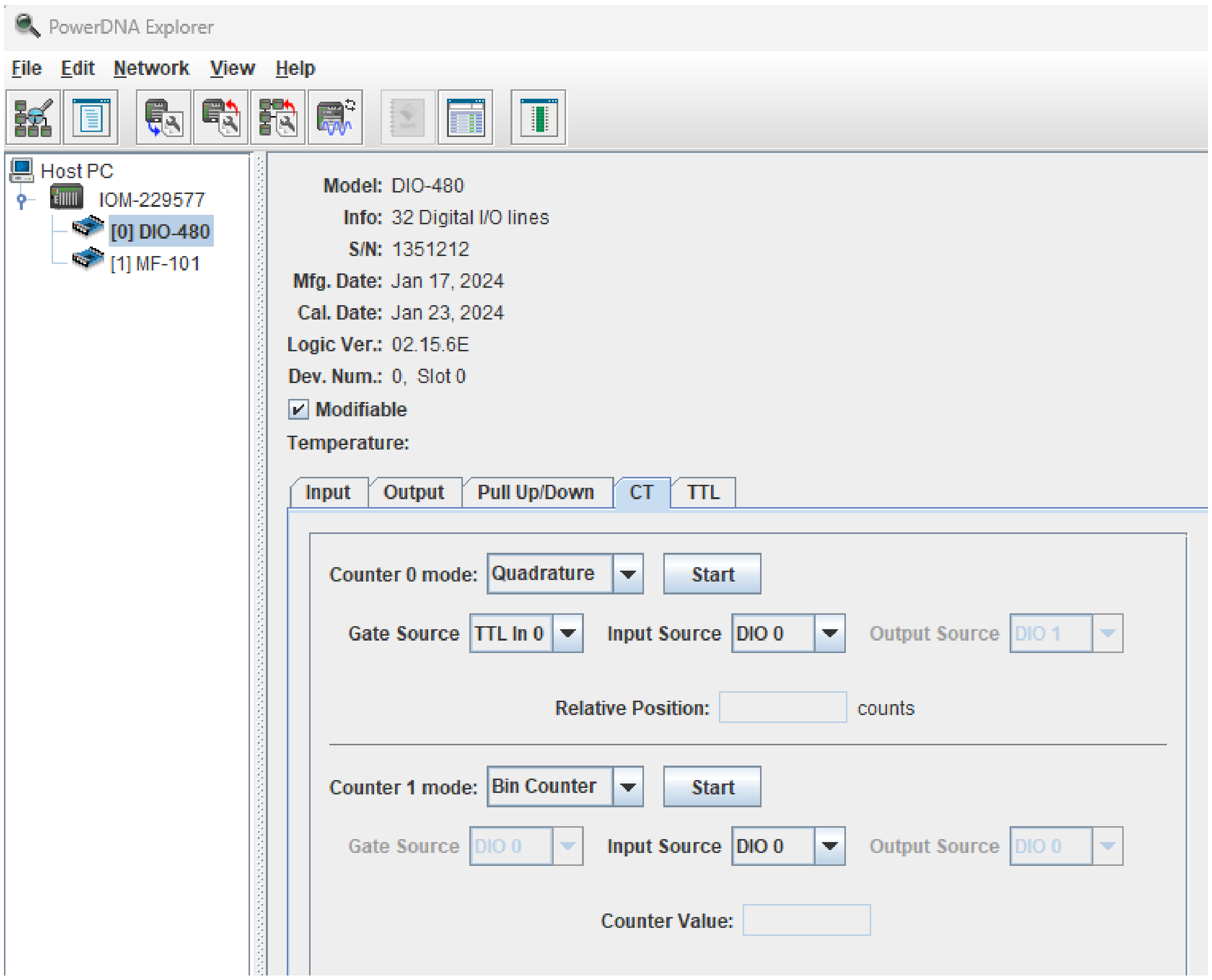
3.5.2 Quadrature Mode

Quadrature Mode counts pulses on the Input Source. The count increases or decreases depending on the Gate Source. Output Source is unused in this mode.

Selection of Quadrature mode is shown on the CT tab for Counter 0 mode in **Figure 3-6**.

Click **Store Configuration** to write settings to hardware.

After you **Start** the counter, data is returned in the **Relative Position** field. This represents the number of pulses on the Input Source in hexadecimal format. Data starts from 0xffffffff and counts up if the value from Gate Source=1. The data counts down if the value from Gate Source=0.



Model: DIO-480
Info: 32 Digital I/O lines
S/N: 1351212
Mfg. Date: Jan 17, 2024
Cal. Date: Jan 23, 2024
Logic Ver.: 02.15.6E
Dev. Num.: 0, Slot 0
Modifiable [✓]
Temperature:

Counter 0 mode: Quadrature
Gate Source: TTL In 0
Input Source: DIO 0
Output Source: DIO 1
Relative Position: counts

Counter 1 mode: Bin Counter
Gate Source: DIO 0
Input Source: DIO 0
Output Source: DIO 0
Counter Value:

Figure 3-6 PowerDNA Explorer CT Tab, Quadrature and Bin Counter Modes

3.5.3 Bin Counter Mode

Bin Counter Mode counts pulses on the Input Source over a 1 second interval. Gate Source and Output Source are unused in this mode.

Selection of Bin Counter mode is shown on the CT tab for Counter 1 mode in **Figure 3-6**.

Click **Store Configuration** to write settings to hardware.

After you **Start** the counter, data is returned in the **Counter Value** field. This represents the number of pulses over a 1 second time interval.

3.5.4 PWM Output Mode

PWM Output Mode generates a square wave on the Output Source. Gate and Input Sources are unused in this mode.

Selection of PWM Output mode is shown on the CT tab for Counter 0 mode in **Figure 3-7**. The following settings are available for PWM Output mode.

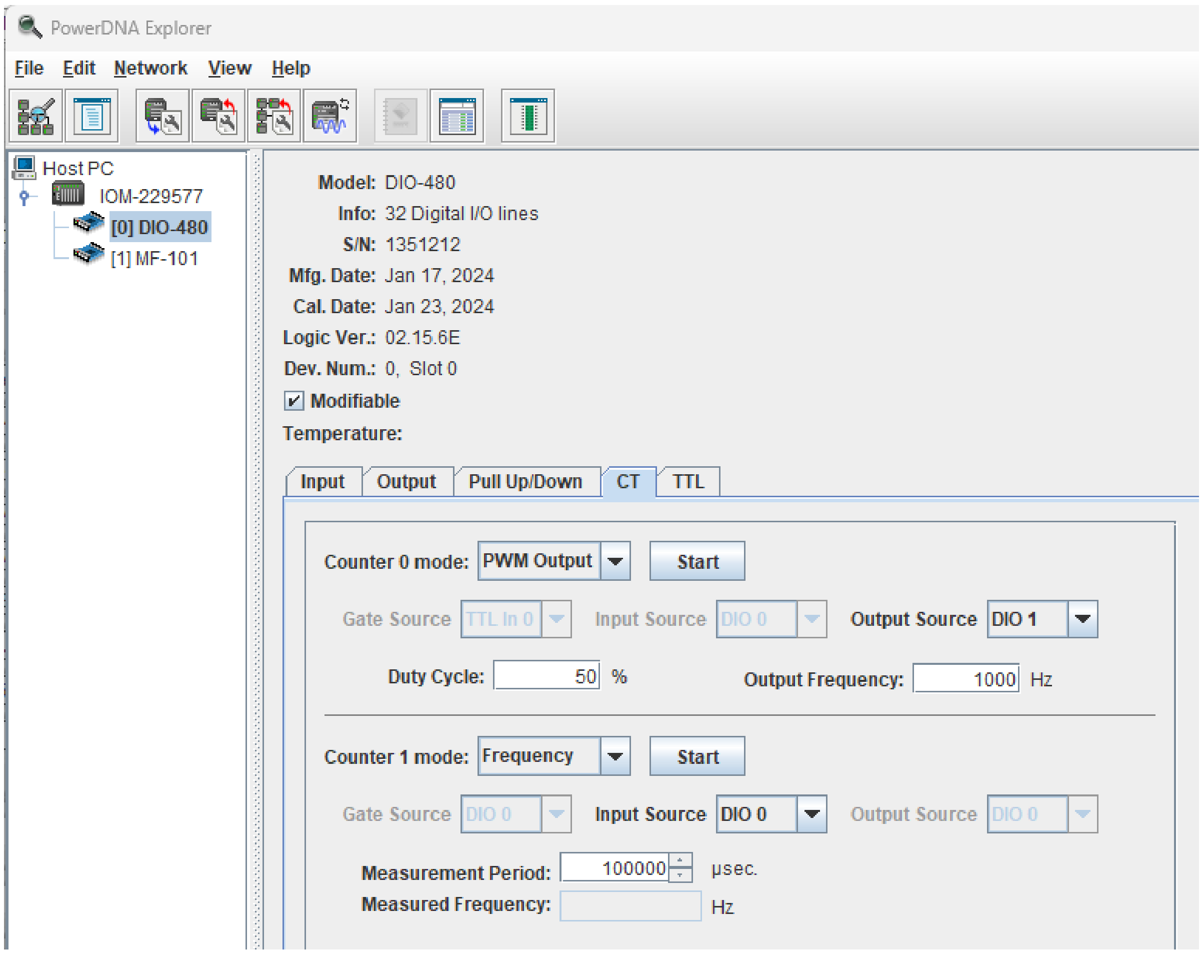
- **Output Source**: Selects an industrial DIO or TTL line for routing the PWM Output. This determines the Output Source displayed on the TTL Output subtab (one of the counter timers or Direct Writes if a DIO line is selected). See Section 3.6.2.

- **Duty Cycle**: Sets the duty cycle of the Output square wave.
- **Output Frequency**: Sets the desired frequency of the Output square wave, between 1 and 25 kHz.

Press the **Enter** key after typing numerical inputs and click **Store Configuration** to write settings to hardware.

After you **Start** the counter, PWM is output using to the selected settings.

NOTE: For FET-based digital outputs, it is easier to generate PWM signals directly through the DO subsystem (Section 3.3).



Model: DIO-480
Info: 32 Digital I/O lines
S/N: 1351212
Mfg. Date: Jan 17, 2024
Cal. Date: Jan 23, 2024
Logic Ver.: 02.15.6E
Dev. Num.: 0, Slot 0
☑ Modifiable
Temperature:

Counter 0 mode: PWM Output
Gate Source TTL In 0 Input Source DIO 0 Output Source DIO 1
Duty Cycle: 50 % Output Frequency: 1000 Hz

Counter 1 mode: Frequency
Gate Source DIO 0 Input Source DIO 0 Output Source DIO 0
Measurement Period: 100000 µsec.
Measured Frequency: Hz

Figure 3-7 PowerDNA Explorer CT Tab, PWM Output and Frequency Modes

3.5.5 Frequency Mode

Frequency Mode measures the frequency of the Input Source over a user-configured time interval. Gate Source and Output Source are unused in Frequency Mode.

Frequency Mode includes the following setting (shown in **Figure 3-7**):

- **Measurement Period**: Time interval for the frequency measurement, between 1 and 32,537,631 microseconds.

Press the Enter key after typing numerical inputs and click **Store Configuration** to write the settings to hardware.

After you **Start** the counter, the measured input frequency in Hz is returned in the **Measured Frequency** field.

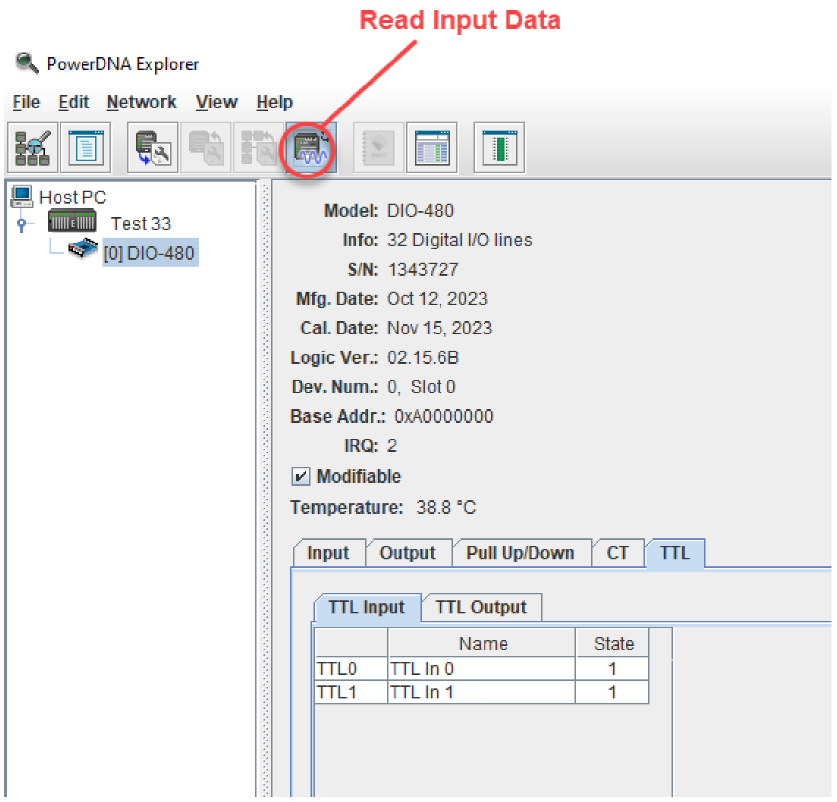
3.6 TTL-Level DIO

To explore the TTL-level digital I/O subsystem, open the TTL tab (**Figure 3-8**).

3.6.1 TTL Input

Click the **Read Input Data** button to read the state of the two TTL input channels. The TTL Input subtab (**Figure 3-8**) contains the following columns:

- **TTLx**: read-only display of the channel number.
- **Name**: read-only display of the channel name.
- **State**: displays the logic state of the channel



Model: DIO-480
Info: 32 Digital I/O lines
S/N: 1343727
Mfg. Date: Oct 12, 2023
Cal. Date: Nov 15, 2023
Logic Ver.: 02.15.6B
Dev. Num.: 0, Slot 0
Base Addr.: 0xA0000000
IRQ: 2
☑ Modifiable
Temperature: 38.8 °C

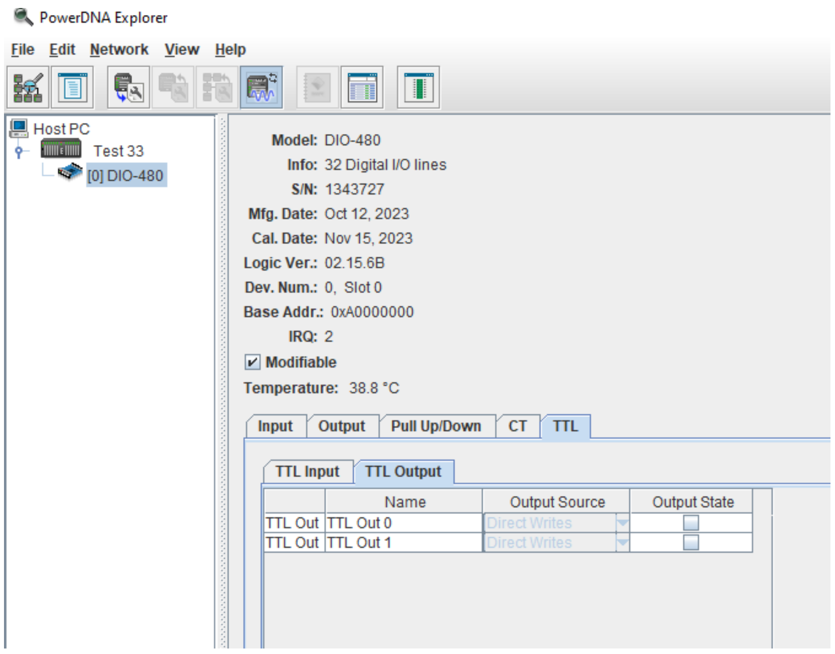
	Name	State
TTL0	TTL In 0	1
TTL1	TTL In 1	1

Figure 3-8 PowerDNA Explorer TTL Tab, TTL Input Subtab

3.6.2 TTL Output

The TTL Output subtab (**Figure 3-9**) contains the following columns:

- **TTLx**: read-only display of the channel number.
- **Name**: read-only display of the channel name.
- **Output Source**: read-only display of the Output Source to be routed to the TTL output. This will be one of the following values and is controlled by the Output Source selection made on the CT tab (Section 3.5.4).
  - **Direct Writes:** This will be displayed as the Output Source if a counter is not set as an output source for the TTL channel. Output State can then be used to write 0 or 1 to the TTL output.
  - **Counter Timer 0:** Displayed when Counter 0 is routed to the TTL output. Output State is not used.
  - **Counter Timer 1:** Displayed when Counter 1 is routed to the TTL output. Output State is not used.
- **Output State**: toggles the logic state of the channel when Direct Writes is the Output Source. Click **Store Configuration** for the changes to take effect.



Model: DIO-480
Info: 32 Digital I/O lines
S/N: 1343727
Mfg. Date: Oct 12, 2023
Cal. Date: Nov 15, 2023
Logic Ver.: 02.15.6B
Dev. Num.: 0, Slot 0
Base Addr.: 0xA0000000
IRQ: 2
✓ Modifiable
Temperature: 38.8 °C

	Name	Output Source	Output State
TTL Out	TTL Out 0	Direct Writes	☐
TTL Out	TTL Out 1	Direct Writes	☐

Figure 3-9 PowerDNA Explorer TTL Tab, Output Subtab

Chapter 4 Programming with the High-level API

This chapter provides the following information about programming the DNx-DIO-480 using the UeiDaq Framework API:

- About the High-level API (Section 4.1)
- Example Code (Section 4.2)
- Create a Session (Section 4.3)
- Assemble the Resource String (Section 4.4)
- Configure the Timing (Section 4.5)
- Start the Session (Section 4.6)
- Industrial Digital Input Session (Section 4.7)
- Industrial Digital Output Session (Section 4.8)
- TTL Digital Input Session (Section 4.9)
- TTL Digital Output Session (Section 4.10)
- Counter Input Session (Section 4.11)
- Counter Output Session (Section 4.12)
- Diagnostics Session (Section 4.13)
- Stop the Session (Section 4.14)

4.1 About the High-level API

UeiDaq Framework is object oriented and its objects can be manipulated in the same manner from different development environments, such as C++, Python, MATLAB, LabVIEW, and more. The Framework is supported in Linux and Windows (7 and up). It is generally simpler to use compared to the low-level API, and it includes a generic simulation device to assist in software development. Therefore, we recommend that users use the Framework unless unconventional functionality is required. Users programming for a non-Linux or non-Windows operating system should instead use the low-level API (Chapter 5).

For more detail regarding the Framework's architecture, please see the *"UeiDaq Framework User Manual"* located under:

Start » All Programs » UEI

For information on the Framework's classes, structures, and constants, please see the *"UeiDaq Framework Reference Manual"* located under:

Start » All Programs » UEI

4.2 Example Code

UeiDaq Framework is bundled with examples for supported programming languages. The example code is located under:

C:\Program FIles (x86)\UEI\Framework

The examples can be accessed via the Windows Start Menu. For example:

Start » All Programs » UEI » Visual C++ Examples

Unlike the low-level examples, Framework examples are board-agnostic, e.g., the “DigitalIn” example works across all UEI digital input layers.

Each high-level example follows the same basic structure listed in the following steps. Subsystem configuration (Step 3) and reading and writing of data (Step 6) are specific to particular subsystems so that information is presented in sections that are tailored to that subsystem.

1. Create a session (Section 4.3).
2. Assemble a resource string (Section 4.4).
3. Configure the session for a particular device and subsystem (Section 4.7 through Section 4.13).
4. Configure the timing (Section 4.5).
5. Start the session (Section 4.6).
6. Read or write data (Section 4.7 through Section 4.13).
7. Stop the session (Section 4.14).

This chapter presents examples using the C++ API, but the concepts are the same no matter which programming language you use. The *“UeiDaq Framework User Manual”* provides additional information about programming in other languages.

4.3 Create a Session

The session object manages all communications with the DNx-DIO-480. Therefore, the first step is always to create a new session.

```
//create a session object

CUeiSession mySession;
```

NOTE: If you want to use multiple subsystems on the DNx-DIO-480 (for example simultaneous digital input and output), you will need to create a new session for each subsystem. Therefore, example sessions for each subsystem will be given unique names.

4.4 Assemble the Resource String

Each session is dedicated to a specific subsystem within the device. Framework uses a resource string to link the session to the hardware. The resource string syntax is similar to a web URL; it should not have any spaces and is case insensitive.

The typical Framework resource string is formatted as follows:

“<device class>://<IP address>/<device number>/<subsystem><channel list>”

Digital input and output subsystems also require a subsystem number be appended to the subsystem name. In those cases, the subsystem and channel list must be separated with a forward slash.

“<device class>://<IP address>/<device number>/
<subsystem><subsystem number>/<channel list>”

The components of a resource string are as follows:

- *<device class>* - By default, Framework examples open with a generic simulated device. To use the DNx-DIO-480, set the device class to `pdna`.
- *<IP address>* - IP address of the IOM.
- *<device number>* - position of the DNx-DIO-480 within the chassis, relative to the other I/O boards.
- *<subsystem>* - one of the following DNx-DIO-480 subsystems:
  - `Di0`: industrial digital input session to configure all 32 lines (Section 4.7). *<channel list>* should always be 0.
  - `Diline0`: industrial digital input session to configure selected lines (Section 4.7)
  - `Do0`: industrial digital output session to configure all 32 lines (Section 4.8). *<channel list>* should always be 0.
  - `Doline0`: industrial digital output session to configure selected lines (Section 4.8)
  - `Di1`: TTL digital input session (Section 4.9). *<channel list>* should always be 0.
  - `Do1`: TTL digital output session (Section 4.10). *<channel list>* should always be 0.
  - `Ci`: counter input session to count events or measure pulse width and period (Section 4.11)
  - `Co`: counter output session to generate pulses and pulse trains (Section 4.12)
  - `Diag`: diagnostic session to read voltage from DIO ADCs, supply voltage from Vcc groups, and surface temperature of the DNx-DIO-480 (Section 4.13)
- *<channel list>* - desired lines or ports within the selected subsystem, either as a comma-separated list of numbers or a range. If the subsystem name ends in a number, separate the subsystem and channel list with a forward slash.

Example 1

This resource string selects all 32 industrial digital input lines on device 1 at IP address 192.168.100.2:

- `“pdna://192.168.100.2/Dev1/Di0/0”`

Example 2

This resource string selects industrial digital input lines 0 through 3 on device 1 at IP address 192.168.100.2:

- `“pdna://192.168.100.2/Dev1/DiLine0/0:3”`

Example 3

The following resource string selects both TTL inputs on device 1 at IP address 192.168.100.2:

- "`pdna://192.168.100.2/Dev1/Di1/0`"

Refer to Section 4.7 through Section 4.13 for details on configuring the different types of subsystems.

4.5 Configure the Timing

The UeiDaq Framework supports two data acquisition modes, point-by-point and edge detection. Edge detection is only supported for industrial digital inputs. Each of the two modes require a specific timing mode. A session may only be configured for one timing mode. Additional modes are supported by the low-level API (Chapter 5).

4.5.1 Point by Point

Point-by-Point mode transfers one sample at a time to/from each configured channel of the I/O board. The delay between samples is controlled by the host application (e.g., by using a Sleep function), thus limiting the data transfer rate to a maximum of 100 Hz. This mode is also known as immediate mode or simple mode.

```
//configure session to use Point-by-Point DAQ mode
mySession.ConfigureTimingForSimpleIO();
```

Repeat for each session in the application.

4.5.2 Edge Detection Timing

Edge Detection will provide data to a read function only if a change of state was detected. If there was no change of state, a timeout error will occur. The DNx-DIO-480 does not support filtering by rising or falling edges. Data will be returned on any state change.

```
//configure session to use EdgeDetection DAQ mode
mySession.ConfigureTimingForEdgeDetection(UeiDigitalEdgeBoth);
```

4.6 Start the Session

Examples of configuration of DNx-DIO-480 subsystems are presented in the following sections. After a session is configured, the session can be started manually:

```
//Start the session.
mySession.Start();
```

If you don't explicitly start the session, it will start automatically the first time you try to transfer data.

4.7 Industrial Digital Input Session

The session may be configured to access the industrial digital input (`Di0` or `Diline0`) subsystem.

4.7.1 Configure Input Channels

The `CreateDIIndustrialChannel()` method adds FET-based digital input channels, sets their hysteresis thresholds, and programs a debouncer to eliminate glitches and spikes.

NOTE: When configuring DNx-DIO-480 channels as both industrial digital inputs and industrial digital outputs, the inputs must be configured before the outputs.

```
CUeiDIIndustrialChannel* CreateDIIndustrialChannel(std::string resource,
    double lowThreshold, double highThreshold, double minPulseWidth);
```

- `resource` – Resource string specifying the port (Section 4.7.1.1) or the line (Section 4.7.1.2)
- `lowThreshold` – Logic level changes from 1 to 0 when the input voltage falls below the low hysteresis threshold.*
- `highThreshold` – Logic level changes from 0 to 1 when the input voltage rises above the high hysteresis threshold.*
- `minPulseWidth` – Debouncer only allows a state change when the input has remained stable at the new level for this number of milliseconds. Use 0.0 to disable the debouncer. The maximum allowable value for `minPulseWidth` width is 655 ms. If a larger value is passed to this method, a value of 655 ms will be used.

*If the signal is in between the low and high thresholds, the detector maintains the previous logic level.

4.7.1.1 Add a Port

Using `Di0` in the resource string adds the entire industrial digital input port to one channel.

```
//Get pointer to input port (channel index = 0) and configure DIO0:31
//with low threshold=2.0 V, high threshold=5.0 V, and
//debouncing interval=1.0 ms.

CUeiDIIndustrialChannel* diPort = diSession.CreateDIIndustrialChannel
      (“pdna://192.168.100.2/Dev1/Di0/0”,
     2.0, 3.0, 1.0);
```

You can reconfigure individual digital input lines using methods in the `CUeiDIIndustrialChannel` class.

```
//Change DIO7 configuration to low threshold=1.5 V, high threshold=3.5V,
//and debouncing interval=2.0 ms.

diPort->SetLowThreshold(7, 1.5);
diPort->SetHighThreshold(7, 3.5);
diPort->SetMinimumPulseWidth(7, 2.0);
```

4.7.1.2 Add Selected Lines

Alternatively, you can configure a subset of lines by specifying `Diline0` in the resource string and appending the desired line numbers. Note that all digital input channels should be initially configured in a single call to `CreateDIIndustrialChannel()`.
Calling `CreateDIIndustrialChannel()` multiple times on the same session will result in only the channels in the final call being added to the session,

```
//Configure DIO2:3 and DIO4:7, initially with the same hysteresis
//thresholds and debounce interval

CUeiDIIndustrialChannel* diLines = diSession.CreateDIIndustrialChannel
     (“pdna://192.168.100.2/Dev1/Diline0/2:3,7:10”, 2.0, 3.0, 1.0);
```

This will create a number of `CUeiDIIndustrialChannel` instances equal to the number of specified digital input lines. Per-channel configuration can then be performed on the channels. The order of channels is the same order in which the channels appeared in the resource string. Note that the `<line>` parameter when setting channel parameters is always 0 when using the "DiLine" session type. The following example sets the low threshold for each of the digital input lines specified in the resource string above.

```
//Set channel index 0 (line 2 in the resource string) low threshold to 0 V

((CUeiDIIndustrialChannel*)session.GetChannel(0))->SetLowThreshold(0, 0.0);

// Set channel index 1 (line 3 in the resource string) low threshold to 1 V

((CUeiDIIndustrialChannel*)session.GetChannel(1))->SetLowThreshold(0, 1.0);

// Set channel index 2 (line 7 in the resource string) low threshold to 2 V

((CUeiDIIndustrialChannel*)session.GetChannel(2))->SetLowThreshold(0, 2.0);

// Set channel index 3 (line 8 in the resource string) low threshold to 3 V

((CUeiDIIndustrialChannel*)session.GetChannel(3))->SetLowThreshold(0, 3.0);

// Set channel index 4 (line 9 in the resource string) low threshold to 4 V

((CUeiDIIndustrialChannel*)session.GetChannel(4))->SetLowThreshold(0, 4.0);

// Set channel index 5 (line 10 in the resource string) low threshold to 5 V

((CUeiDIIndustrialChannel*)session.GetChannel(5))->SetLowThreshold(0, 5.0);
```

4.7.1.3 Edge Detection

When configuring industrial digital inputs to return data on change-of-state events, an edge mask must be provided to the channel object. Only the first channel's edge mask will be used when configuring the session, regardless of "Di" or "DiLine" session types.

```
//Return data on change-of-state for DI lines 4:31

uInt32 edgeDetectMask = 0xfffffff0;
diPort->>SetEdgeMask(edgeDetectMask);
```

Remember that to receive data on change-of-state events, the timing for the digital input channel must be configured for EdgeDetection mode (Section 4.5.2).

4.7.2 Read Data

Reading data is done using a Digital Reader object. This is created using the session's data stream object.

```
//Create a reader object and link it to the session’s data stream.
CUeiDigitalReader diReader(diSession.GetDataStream());
```

4.7.2.1 Read DI Port

When reading industrial digital input data from a `Di0` session, use `uInt32` data. A single `uInt32` will be returned with the low/high debounced status mask of all 32 channels.

```
//Read state of DIO0:31
uInt32 data;
diReader.ReadSingleScan(data);
```

4.7.2.2 Read Specific DI Lines

When reading industrial digital input data from a `Diline0` session, use `uInt32` data. A number of `uInt32` values will be returned that will be equal to the number of configured channels. Only bit 0 of each 32-bit value should be used (0 is low, 1 is high) .

```
//Read state of DIO0:31
uInt32* digitalState = new uInt32[session.GetNumberOfChannels()];
diReader.ReadSingleScan(digitalState);
```

NOTE: If you are simultaneously running a digital output session, ensure that the output mask is disabled for the input-only lines. Otherwise, the reader will return the values written to the port.

4.7.2.3 Read Change of State Data (Edge Detection)

When reading industrial digital input data on a channel that is configured for Edge Detection, data will only be returned if a change-of-state was detected. Otherwise, a timeout error will occur. For the DNx-DIO-480, a read will return up to three `uint32` words as follows:

data[0] = positive edge status
data[1] = negative edge status
data[2] = optional time stamp.

```
//Read data. Timeout will occur if there was no change-of-state

    uInt32 readData[3];
    for (int i = 0; i < 10; i++) {
        try {
            diReader.ReadSingleScan(readData);
        } catch (CUeiException &e) {
            if (e.GetError() == UEIDAQ_TIMEOUT_ERROR) {
                std::cout << "Timeout waiting for edge" <<std::endl;
                continue;
            }
        }
        // The default resolution for time stamp is 10 microseconds
        // To convert from readData[2] in ticks to seconds:
        // double seconds = readData[2] * (1.0 / (66000000.0 / 660.0))
        //           66 MHz ticks       66 MHz clock / ticks per 10 µs
        std::cout << std::dec << "Time stamp: " << readData[2] << ": ";
        std::cout << std::hex << "Positive Edge: 0x" << readData[0] << ","
            << " Negative Edge: 0x" << readData[1] << std::endl;
        std::this_thread::sleep_for(std::chrono::milliseconds(1000));
    }
```

4.7.3 Read Input Voltages

You can read voltage from the DIO ADCs by creating a separate Diagnostic session (Section 4.13).

4.8 Industrial Digital Output Session

The session may be configured to access the industrial digital output (`Do0` or `Doline0`) subsystem. Because sessions are unidirectional, you will need a dedicated output session even though output and input share the same physical port.

4.8.1 Configure Output Channels

The `CreateDOIndustrialChannel()` method adds FET-based digital output channels and configures PWM on those channels.

NOTE: When configuring DNx-DIO-480 channels as both industrial digital inputs and industrial digital outputs, the inputs must be configured before the outputs.

```
CUeiDOIndustrialChannel* CreateDOIndustrialChannel(std::string resource,
   tUeiDOPWMMode pwmMode, uInt32 pwmLengthUs, uInt32 pwmPeriodUs,
   double pwmDutyCycle);
```

- `resource` – Resource string specifying the port (Section 4.8.1.1) or the line (Section 4.8.1.2)
- `pwmMode` – Type of pulse train to output (Section 4.8.1.4)

- `pwmLengthUs` – Total duration of soft start and/or soft stop pulse train in microseconds; ignored in other PWM modes
- `pwmPeriodUs` – Period in microseconds; min 5 µs, max 254200 µs
- `pwmDutyCycle` – Duty cycle between 0.0 and 1.0; ignored in soft start/stop modes

4.8.1.1 Add a Port

Using `Do0` in the resource string adds the entire digital output port to one channel.

```
//Get pointer to output port (channel index = 0) and configure DIO0:31
//for output with no PWM. The last 3 parameters are
//ignored when PWM is disabled.

CUeiDOIndustrialChannel* doPort = doSession.CreateDOIndustrialChannel
     (“pdna://192.168.100.2/Dev1/Do0”,
     UeiDOPWMDisabled, 0, 0, 0);
```

All outputs in the channel are enabled by default. You can selectively enable/disable outputs with a 32-bit output mask (LSB is DIO0).

NOTE: If you are simultaneously running a digital input session, ensure that the output mask is disabled (i.e., set to 0) for the input-only channels.

```
//Enables output on DIO0:7. DIO8:31 are configured as input-only.

doPort->SetOutputMask(0x000000ff);
```

PWM features are configurable on a line-by-line basis.

```
//Configure DIO5 for a soft start; period = 80 µs and duration = 2000 µs

doPort->SetPWMMode(5, UeiDOPWMSoftStart);
doPort->SetPWMPeriod(5, 80);
doPort->SetPWMLength(5, 2000);
```

4.8.1.2 Add Selected Lines

Alternatively, you can configure a subset of lines by specifying `Doline0` in the resource string and appending the desired line numbers.

```
//Configure DIO2:3 and DIO4:7 with 25% and 50% duty cycles respectively.

doSession.CreateDOIndustrialChannel
   (“pdna://192.168.100.2/Dev1/Doline0/2:3”,
   UeiDOPWMContinuous, 1000, 50, 0.25);
doSession.CreateDOIndustrialChannel
   (“pdna://192.168.100.2/Dev1/Doline0/4:7”,
   UeiDOPWMContinuous, 1000, 50, 0.5);
```

This approach creates one channel per line. Unlike a `Do0` line, each `Doline` is reconfigured using a unique channel index as follows:

```
//Get pointer to DIO4. DIO4 is ch2 (the third channel) in the
//list created above.

CUeiDOIndustrialChannel* dochannel =
     dynamic_cast<CUeiDOIndustrialChannel*>(doSession.GetChannel(3));

//Set DIO4 period to 200 µs (pass in 0 for the line parameter)

dochannel->SetPWMPeriod(0, 200);
```

Even if you configured only a subset of lines, the output mask applies to all 32 lines. You can use the same output mask code shown in Section 4.8.1.1. Call `SetOutputMask()` using the first channel created. Using other channels to set the output mask will result in the output mask being ignored.

4.8.1.3 Configure Pull-up/down Resistors

You can connect a DIO line to Vcc and/or Gnd (Figure 2-2).

- `UeiDigitalTerminationNone` – no termination
- `UeiDigitalTerminationPullUp` – enable only pull-up resistor
- `UeiDigitalTerminationPullDown` – enable only pull-down resistor
- `UeiDigitalTerminationPullUpPullDown` – enable both pull-up and pull-down resistor

```
//Connect pull-up resistor between DIO1 and Vcc.

doPort->SetTermination(1, UeiDigitalTerminationPullUp);
```

4.8.1.4 PWM Modes

Choose one of the following options for the `pwmMode` input parameter:

- `UeiDOPWMDisabled` – disable PWM
- `UeiDOPWMSoftStart` – generate a pulse train after writing 1 if its previous state was 0. The PWM duty cycle gradually increases from 0% to `pwmDutyCycle` over `pwmLengthUs`.
- `UeiDOPWMSoftStop` – generate a pulse train after writing 0 if its previous state was 1. The PWM duty cycle gradually decreases from `pwmDutyCycle` to 0% over `pwmLengthUs`.
- `UeiDOPWMSoftBoth` – generate a pulse train for both a low-to-high and high-to-low transition.
- `UeiDOPWMContinuous` – continuously generates a pulse train with `pwmDutyCycle.` When writing to digital outputs, ensure that a 1 is written to any output that is configured for `UeiDOPWMContinuous` mode.
- `UeiDOPWMGated` – generates a pulse train with `pwmDutyCycle` only when a 1 is written to the output.

4.8.1.5 Configure PWM Push/Pull

You can specify which FETs are switched by the PWM output:

- `UeiDOPWMOutputPush` – switch only high-side FET
- `UeiDOPWMOutputPull` – switch only low-side FET
- `UeiDOPWMOutputPushPull` – switch both FETs. This is the default PWM output mode.
- `UeiDOPWMOutputOff` – no PWM applied to either FET

```
//Enable PWM on only high-side FET of DIO1.

doPort->SetPWMOutputMode(1, UeiDOPWMOutputPush);
```

4.8.1.6 Configure H-Bridge

Once a digital output object is created, industrial DIO channels that are configured for output can also be configured for H-Bridge.

```
//Enables output on DIO0:1 and DIO24:25.
//The other bits are configured as input-only.

doPort->SetOutputMask(0x03000003);
```

The digital output channel has an `HBridgeMask` property. Only the HBridge mask of the first configured digital output channel is used to configure the DNx-DIO-480 H-Bridge. The bits in the HBridge mask represent the *channel pair* so DIO lines 0 and 1 correspond to bit 0 of the HBridge mask and DIO lines 24 and 25 correspond to bit 12 of the HBridge mask (divide the DIO line by 2 and round down to get the HBridge mask bit number).

```
//Set the HBridgeMask for channels DIO0:1 and DIO24:25.

doPort->SetHBridgeMask(0x00001001);
```

4.8.1.7 Circuit Breakers

The DNx-DIO-480 outputs are protected by circuit breakers that can be configured through software. The circuit breakers can be enabled and configured by using the `CUeiDOProtectedChannel` class instead of the `CUeiDOIndustrialChannel` class.

```
CUeiDOProtectedChannel* CreateDOProtectedChannel(std::string resource,
           double underCurrentLimit, double overCurrentLimit,
           double currentSampleRate, bool autoRetry, double retryRate);
```

NOTE: `CUeiDOProtectedChannel` inherits `CUeiDOIndustrialChannel` To set the class members corresponding to the arguments that would normally be passed to `CreateDOIndustrialChannel()`, use the set methods for `CUeiDOIndustrialChannel` (Section 4.8.1).

When using the `CUeiDOProtectedChannel` class with the DNx-DIO-480, only the ability to enable/disable circuit breakers and set the over/under count are needed. Subsequently, arguments passed to `CreateDOProtectedChannel()` are ignored and class members and methods unrelated to that functionality are unused.

Create a DO protected channel as follows. Default values are provided for the arguments that are unused with the DNx-DIO-480.

```
//Get pointer to output port (channel index = 0)

CUeiDOProtectedChannel* doPort = doSession.CreateDOProtectedChannel
     (“pdna://192.168.100.2/Dev1/Do0”,
     -0.1, 0.1, 100.0, false, 10.0);
```

All outputs in the channel are enabled by default. You can selectively enable/disable outputs with a 32-bit output mask (LSB is DIO0). The circuit breakers are also enabled by default.

Once a digital output protected channel object is created, the circuit breakers can be configured by enabling circuit breakers and specifying the number of consecutive out of range samples need to occur before a breaker is tripped. Samples occur every 10 µs. The minimum number of samples that can be specified is 1500 and the maximum is 262,143 (18 bits).

```
//Enable circuit breaker on DO line 12

doPort->EnableCircuitBreaker(12, true);

//Set the count of how many consecutive out of range samples will result
//in tripping the circuit breaker. This applies to DO lines on the port
//with enabled circuit breakers. Samples occur every 10 µs.

doPort->SetOverUnderCount(1500);
```

A `CUeiCircuitBreaker` object can be used to read back the status of the circuit breakers and also to reset the circuit breakers. The object is created the same way as a reader or writer object.

```
//Create the CUeiCircuitBreaker object and link it the session’s
//data stream.
//Pass in 0 for the unused port argument.

CUeiCircuitBreaker circuitBreaker(doSession.GetDataStream(), 0);
```

Once the session is started the status can be read using `ReadStatus`. Only the current status is supported. The sticky status is unused.

```
//Read circuit breaker status

uInt32 cbStatus;
circuitBreaker.ReadStatus(&cbStatus, NULL);
```

To reset the circuit breaker the, `Reset` method can be used. The parameter passed to this method is a bitmask that selects which channels to reset. A 1 in a channel's bit position will reset the circuit breaker on that channel.

```
//Reset the circuit breakers on the DO lines 0-15

uInt32 reset_channel_mask = 0x0000FFFF;
circuitBreaker.Reset(reset_channel_mask);
```

4.8.2 Write Data

Writing data is done using a Digital Writer object.

```
//Create a writer object and link it to the session’s data stream.

CUeiDigitalWriter doWriter(doSession.GetDataStream());
```

4.8.2.1 Write Digital Output

Digital data is written as a 32-bit integer. The writer updates all lines in the port, even if `Doline` configured only a subset of lines. FET-based outputs should be enabled using `SetOutputMask()`. Data for bits that are not enabled will be ignored.

```
//Write a 1 on DIO15:8 and a 0 on all other DIO lines.

uInt32 data = 0x0000ff00;
doWriter.WriteSingleScan(&data);
```

4.8.2.2 Write PWM Configuration

The `CUeiDigitalWriter` class can also be used to change PWM configuration while a session is running. Update PWM parameters through the channel object and then use the writer to apply the new settings through `WritePwmUpdate()`.

```
//Update channel 2's duty-cycle to 25%.
//This is not applied to the output yet
doPort->SetPWMDutyCycle(2, 0.25);

//This call will apply the new PWM configuration
doWriter.WritePwmUpdate();
```

4.8.3 Read Output Voltages

You can monitor digital outputs using the `diag` subsystem, as described in Section 4.13.

4.9 TTL Digital Input Session

The session may be configured to access the TTL digital input (`Di1`) subsystem.

4.9.1 Configure Input Port

The DNx-DIO-480 has two TTL input lines. In the resource string, specify port 0 as shown in the code below. The TTL input port includes both TTL input lines. Unlike an industrial digital session, you cannot configure a TTL session to only access a subset of lines.

```
//Configure session to read the TTL input port.

ttliSession.CreateDIChannel(“pdna://192.168.100.2/Dev1/Di1/0”);
```

4.9.2 Read Data

Reading data is done using a Digital Reader object `(CUeiDigitalReader)`. Digital data is stored in a 16-bit integer buffer. Bits 0 and1 are TTL lines. The other bits are currently reserved.

```
//Create a reader object and link it to the session’s data stream.

CUeiDigitalReader diReader(ttliSession.GetDataStream());

//Read state of all lines in the port. A scan returns a 16-bit integer.

uInt16 data[1];
diReader.ReadSingleScan(data);
```

4.10 TTL Digital Output Session

The session may be configured to access the TTL digital output (`Do1`) subsystem.

4.10.1 Configure Output Port

The DNx-DIO-480 has two TTL output lines. The resource string should specify port 0 as shown in the example below. The TTL output port includes both TTL output lines.

```
//Configure session to use the TTL output port.

ttloSession.CreateDOChannel("pdna://192.168.100.2/Dev1/Do1/0");

//Create a writer object and link it to the session's data stream.

CUeiDigitalWriter diWriter(ttliSession.GetDataStream());
```

4.10.2 Write Data

Writing data is done using a Digital Writer object. Digital data is written as a 16-bit integer: Bits 0 and 1 are the TTL lines. The other bits are unused.

```
//Create a writer object and link it to the session's data stream.

CUeiDigitalWriter doWriter(ttloSession.GetDataStream());

// writeData = 0, Writes both TTL outputs low
// writeData = 1, Writes TTL output 0 high, TTL output 1 low
// writeData = 2, Writes TTL output 0 low, TTL output 1 high
// writeData = 3, Writes both TTL outputs high
// Set both TTL lines high

uInt16 data = 3;
doWriter.WriteSingleScan(&data);
```

4.11 Counter Input Session

The session may be configured to access the counter input (`Ci`) subsystem.

4.11.1 Add Input Channels

The `CreateCIChannel()` method adds counter input channels and sets basic configuration parameters.

```
CUeiCIChannel* CreateCIChannel(std::string resource,
   tUeiCounterSource source, tUeiCounterMode mode,
   tUeiCounterGate gate, Int32 divider, Int32 inverted);
```

- `resource` – Resource string for counter 0 or counter 1
- `source` – Set CLKIN to either the internal 66 MHz clock or an external input pin (Section 4.11.2)
- `mode` – Counting mode (Section 4.11.3)
- `gate` – Use either an external or a software gate to enable the counter
- `divider` – Prescaler divides source signal by this factor; default = 1
- `inverted` – TRUE to invert source signal

```
//Configure counter 0 to count events on an external pin.
//An internal gate starts the count immediately.
//Source is divided by 2 and not inverted.

ciSession.CreateCIChannel(“pdna://192.168.100.2/Dev1/Ci0”,
   UeiCounterSourceInput, UeiCounterModeCountEvents,
   UeiCounterGateInternal, 2, false);
```

4.11.2 Route Counter to DIO Pins

The counter’s CLKIN, GATE, and CLKOUT lines can be internally routed to the following pins:

- **fetX** - Industrial DIO pins, e.g., “fet3” for DIO 3
- **ttlX** - TTL DIO pins, e.g., “ttl1” for TTL1

The external CLKIN pin is only used when the counter is configured with `source = UeiCounterSourceInput`. Similarly, the GATE pin is only used when the counter is configured with `gate = UeiCounterGateExternal`.

```
//Obtain pointer to the input channel (only one channel in this case).

CUeiCIChannel* counter =
   dynamic_cast<CUeiCIChannel*>(ciSession.GetChannel(0));

//Route CLKIN to DIO1.
//Route GATE to TTL1.
//Route CLKOUT to DIO2, DIO3, and TTL0.

counter->SetSourcePin(“fet1”);
counter->SetGatePin(“ttl1”);
counter->SetOutputPins(“fet2, fet3, ttl0”);
```

You can set up an optional input debouncer for CLKIN and GATE. The maximum allowable value for the minimum pulse width is 7.94 ms. If a larger value is passed to either of these methods, a value of 7.94 ms will be used.

```
//Allow state change only when inputs have stayed stable for 1.0 ms.

counter->SetMinimumSourcePulseWidth(1.0);
counter->SetMinimumGatePulseWidth(1.0);
```

For fetX inputs, you must also create a separate industrial digital input session (Section 4.7). This configures and starts up the A/D converter.

```
//Create new session.

CUeiSession diSession;

//Configure session to read from FET-based digital inputs.
//Low threshold = 2.0 V, high threshold = 3.0 V,
//debouncer interval = 1.0 ms

diSession.CreateDIIndustrialChannel(“pdna://192.168.100.2/Dev1/Di0”,
   2.0, 3.0, 1.0);

//Configure timing for Point by Point DAQ mode.

diSession.ConfigureTimingForSimpleIO();
```

You do not need a separate session for TTL-level inputs (ttlX, syncX), nor do you need one for outputs. If you are simultaneously running a digital output session and want to read in external inputs, remember to disable the output mask on input-only lines. The counter session automatically overrides digital output session settings on output lines.

NOTE: CLKOUT should always be routed to an external pin, even if the counter is only used for input. CLKOUT remains high during a counter input session.

4.11.3 Counter Input Modes

Choose one of the following options for the `mode` parameter:

- `UeiCounterModeCountEvents` – Count pulses on an external pin, or use as a timer by counting internal clock cycles.
- `UeiCounterModeBinCounting` – Count pulses over a user-specified time interval (Section 4.11.3.1).
- `UeiCounterModeMeasurePulseWidth` – Count the number of 66 MHz clocks while the input signal is high.
- `UeiCounterModeMeasurePeriod` – Count the number of 66 MHz clock ticks over the specified number of periods (Section 4.11.3.2). The number of clock ticks returned will actually have occurred over the specified number of periods plus 1, e.g., if 10 periods are specified, then the returned number of clock ticks will have occurred over 11 periods.
- `UeiCounterModeTimedPeriodMeasurement` – Measure the average period over a user-specified time interval (Section 4.11.3.1); period is returned as the number of 66 MHz clock ticks.

- `UeiCounterModeQuadratureEncoder` – Quadrature encoder measurement; PWM signal on GATE controls the count direction
- `UeiCounterModeDirectionCounter` – Count up if GATE is high and count down if GATE is low

4.11.3.1 Set Capture Time Interval

The time interval for `UeiCounterModeBinCounting` and `UeiCounterModeTimedPeriodMeasurement` is configured using the session's timing object.

```
//Get pointer to session’s timing object.
CUeiTiming* ciTiming = ciSession.GetTiming();
//Set frequency to 0.5 Hz; count is returned every 2.0 sec.
ciTiming->SetScanClockRate(0.5);
```

4.11.3.2 Set Number of Periods

In `UeiCounterModeMeasurePeriod` mode, the counter can be configured to measure the total duration of N+1 periods.

```
//Update the counter when 11 (N+1) periods have elapsed.
counter->SetPeriodCount(10);
```

The counter returns the previous measurement until the specified number of periods have been counted again.

4.11.4 Read Count Data

Reading data is done using a reader object. Digital data is stored in a 32-bit integer buffer.

```
//Create a reader object and link it to the session’s data stream.
CUeiCounterReader ciReader(ciSession.GetDataStream());
//Read the current count value.
uInt32 data[1];
ciReader.ReadSingleScan(data);
```

4.12 Counter Output Session

The session may be configured to access the counter output (`Co`) subsystem.

4.12.1 Add Output Channels

The `CreateCOChannel()` method adds counter output channels and configures the shape of the output signal.

```
CUeiCOChannel* CreateCOChannel(std::string resource,
   tUeiCounterSource source, tUeiCounterMode mode,
   tUeiCounterGate gate, uInt32 tick1, uInt32 tick2,
   Int32 divider, Int32 inverted);
```

- `resource` – Resource string for counter 0 or counter 1
- `source` – Set CLKIN to either the internal 66 MHz clock or an external input pin (Section 4.11.2)
- `mode` – Counting mode (Section 4.12.3)
- `gate` – Use either an external or a software gate to enable the counter
- `tick1` – Number of counts for which output is low
- `tick2` – Number of counts for which output is high
- `divider` – Prescaler divides source signal by this factor; default = 1
- `inverted` – TRUE to invert source signal

```
//Configure counter 0 to output pulse train (period=6ms, duty cycle=75%).
//Count ticks of an undivided, uninverted 66 MHz source clock.
//An internal gate starts the output immediately.

coSession.CreateCOChannel(“pdna://192.168.100.2/Dev1/Co0”,
   UeiCounterSourceClock, UeiCounterModeGeneratePulseTrain,
   UeiCounterGateInternal, 100000, 300000,
   1, false)
```

4.12.2 Route Counter to DIO Pins

Refer to Section 4.11.2 and the methods in the `CUeiCOChannel` class.

4.12.3 Counter Output Modes

Choose one of the following options for the `mode` parameter:

- `UeiCounterModeGeneratePulse` – Generate a single pulse
- `UeiCounterModeGeneratePulseTrain` – Generate a continuous pulse train
- `UeiCounterModePulseWidthModulation` – Generate a pulse width modulated waveform (same as `GeneratePulseTrain`)

4.12.4 Write Output Parameters

You can write new `tick1` and `tick2` values to the counter using a writer object. This is used to change the PWM period and/or duty cycle after the session has already been started.

```
//Create a writer object and link it to the session's data stream.

CUeiCounterWriter coWriter(coSession.GetDataStream());

//Buffer must be large enough to contain two 32-bit integers per channel.

uInt32 data[2]={20000, 5000};

//Set tick1 = 20000 (low duration)
//Set tick2 = 5000 (high duration)

coWriter.WriteSingleScan(data);
```

4.13 Diagnostics Session

A diagnostic session may be configured to use the `diag` subsystem to read voltage from the DIO ADCs, supply voltage from Vcc groups, and the surface temperature of the DNx-DIO-480.

4.13.1 Add Input Channels

The `CreateDiagnosticChannel()` method adds the diagnostic channels specified in the resource string. The `Diag` subsystem supports the channel numbers listed in Table 4-1.

```
//Configure session to read all diagnostics.

diagSession.CreateDiagnosticChannel(“pdna://192.168.100.2/Dev1/
diag0:40”);
```

Table 4-1 Diagnostic Channel Numbers

Channel #	Description
0	ADC Voltage on DIO0
1	ADC Voltage on DIO1
:	:
30	ADC Voltage on DIO30
31	ADC Voltage on DIO31
32	ADC Voltage Reading for Channel Group 0
33	ADC Voltage Reading for Channel Group 1
:	:
38	ADC Voltage Reading for Channel Group 6
39	ADC Voltage Reading for Channel Group 7
40	DIO-480 temperature reading

To help keep track of the different channels in a session, you can retrieve an abbreviated description of each channel with `GetAliasName()`. The following example code returns the string “adc_fet0” when used with the channel list created above.

```
//Retrieve name of first channel in the CreateDiagnosticChannel() list.

diagSession.GetChannel(0)->GetAliasName();
```

4.13.2 Read Data

Read diagnostic data using a `CUeiAnalogScaledReader` instance. The following example code prints that alias name for all the diagnostic channels as well as the data read.

```
//Create a reader object and link it to the session’s data stream.

CUeiAnalogScaledReader diagReader(diagSession.GetDataStream());

//Buffer must be large enough to contain diagnostic data for all channels

double buffer[41];

//Get data

diagReader.ReadSingleScan(buffer);

// Print channel alias name and read data

for (int ch = 0; ch < session.GetNumberOfChannels(); ch++) {
    std::cout << "[" << ch << "] " << "Name: " <<
    diagSession.GetChannel(ch)->GetAliasName() << " "
    << buffer[ch] << std::endl;
}
```

4.14 Stop the Session

The session will automatically stop and clean itself up when the session object goes out of scope or when it is destroyed. To manually stop the session:

```
//Stop the session.

mySession.Stop();
```

To reuse the object with a different set of channels or parameters, you can manually clean up the session as follows:

```
//clean up session and free resources

mySession.CleanUp();
```

Chapter 5 Programming with the Low-level API

This chapter provides the following information about programming the DNx-DIO-480 using the low-level API:

- About the Low-level API (Section 5.1)
- Example Code (Section 5.2)
- Data Acquisition Modes (Section 5.3)
- Point-by-Point API (Section 5.4)
- Async Events API (Section 5.5)
- RtDMap API (Section 5.6)
- RtVMap API (Section 5.7)
- AVMap API (Section 5.8)
- H-Bridge Configuration (Section 5.9)
- Circuit Breaker Configuration (Section 5.10)
- Guardian Diagnostics (Section 5.11)

5.1 About the Low-level API

The low-level API provides direct access to the DAQBIOS protocol structure and registers in C. The low-level API is intended for speed-optimization, when programming unconventional functionality, or when programming under Linux or real-time operating systems.

When programming in Windows OS, we recommend that you use the UeiDaq High-level Framework API (see Chapter 4). The Framework simplifies the low-level API, making programming easier and faster while still providing access to the majority of low-level API features. Additionally the Framework supports a variety of programming languages and the use of scientific software packages such as LabVIEW and MATLAB.

For additional information regarding low-level programming, refer to the “PowerDNA API Reference Manual” located in the following directories:

- On Linux: *<PowerDNA-x.y.z>/docs*
- On Windows: *C:\Program Files (x86)\UEI\PowerDNA\Documentation*

NOTE: The DNx-DIO-480 is supported in PowerDNA version 5.2.0.8+. If you’re unsure if your version supports the board please contact Technical Support at uei.support@ametek.com.

5.2 Example Code

Application developers are encouraged to explore the self-documented source code examples to get started programming UEI products. The example code is located in the following directories:

- On Linux: *<PowerDNA-x.y.z>/src/DAQLib_Samples*
- On Windows:
  C:\Program Files (x86)\UEI\PowerDNA\SDK\Examples\Visual C++

The I/O board number is embedded in the name of the example code. For example, the *Sample480* folder contains example code specific to the DNx-DIO-480. The example code should run out of the box after inputting the IOM's IP address and the board's Device Number (DEVN).

5.3 Data Acquisition Modes

Table 5-1 lists the data acquisition (DAQ) modes available for transferring data between the DNx-DIO-480 and the low-level user application.

Table 5-1 DAQ Modes supported by Low-level API

DAQ Mode	DIn	DOut	TTL	CT
Point-by-Point	●	●	●	●
ACB				
RtDMap	●	●	●	●
RtVMap	●			
ADMap				
AVMap	●			

- **Point-by-Point:** Transfers one data point at a time to/from each configured channel of a single I/O board. Timing is controlled by the user application, which limits the transfer rate to 100 Hz. Point-by-Point mode is also known as immediate mode or simple mode.
- **Real-Time Data Map (RtDMap):** Transfers a packet containing one data point for each channel in the user-defined map. The newest data is transferred and old data is discarded. RtDMap is designed for closed-loop (control) applications and may include channels across multiple I/O boards.
- **Real-Time Variable Map (RtVMap):** Transfers a packet containing a variable number of data points per channel. RtVMap buffers the data and transfers the oldest data first. RtVMap is designed for closed-loop (control) applications and may include channels across multiple I/O boards. With RtVMap, the user application controls the timebase and initiates the request for data delivery.
- **Asynchronous Variable Map (AVMap):** Transfers a packet containing a variable number of data points per channel. AVMap buffers the data and transfers the oldest data first. AVMap is designed for closed-loop (control) applications and may include channels across multiple I/O boards. With AVMap, a hardware condition, e.g., a timer countdown, triggers data delivery.

ACB and ADMap are currently not supported on the DNx-DIO-480.

Please refer to *"FAQ - Data Acquisition Modes"* for an overview and comparison of all the different acquisition modes offered by UEI. The *"PowerDNx Protocol Manual"* includes more detailed information about the protocols. Both of these documents are located in the directories listed in Section 5.1.

NOTE: Multiple subsystems (DIn, DOut, etc.) may be used together as long as they share the same DAQ mode. It is not possible to mix and match multiple DAQ modes on a single IO board.

5.3.1 Async Events Mode

The DNx-DIO-480 supports asynchronous event handling. This event-driven mode runs in a separate thread alongside the selected DAQ mode. The firmware sends an event packet when a specific event occurs. Events on the DNx-DIO-480 include:

- DIn periodic status
- DIn pin change of state

You can call any of the DAQ mode functions upon receiving the event.

5.4 Point-by-Point API

This section summarizes the low-level API used to configure, read from, and write to the DNx-DIO-480 in Point-by-Point DAQ mode. The functions and parameters are described in detail in the *“PowerDNA API Reference Manual”*. Please see *DIO\Sample480* under the *Examples* folder referenced above for a comprehensive example which includes typical initialization, error handling, and usage of these functions. The example splits the I/O subsystems into separate cases, making it easy to copy-paste different subsystems into a true multifunction application.

The information in this section is intended as a supplement to the example code and the *“PowerDNA API Reference Manual”*.

5.4.1 Digital I/O

Table 5-2 lists the low-level API for the DNx-DIO-480 digital I/O subsystems..

Table 5-2 Low-level Digital I/O API

	Function	Description
Industrial DIn	`DqAdv480DIORead`	Read the current and debounced states on industrial DIO lines.
	`DqAdv480DIOSetDebouncer`	Set debouncing interval for industrial digital inputs.
	`DqAdv480DIOSetLevels`	Set low and high voltage levels for industrial digital inputs.
Industrial DOut	`DqAdv480DIOReadLastWrite`	Read back the last state written to industrial digital outputs.
	`DqAdv480DIOSetSource`	Set industrial DIO channel output source.
	`DqAdv480DIOSetOutputMode`	Set industrial DIO channel output mode.
	`DqAdv480DIOSetHBridge`	Set industrial DIO H-Bridge configuration.
	`DqAdv480DIOSetPWM`	Configure PWM mode on digital outputs.
	`DqAdv480DIOSetTermination`	Configure pull up/down resistors.
	`DqAdv480DIOWrite`	Set digital output state to 0 or 1.
TTL DIO	`DqAdv480TTLRead`	Read the state of the TTL lines.
	`DqAdv480TTLSetSource`	Sert source for a TTL output.
	`DqAdv480TTLWrite`	Set state of TTL outputs.

5.4.2 Counters

Table 5-3 lists the low-level API for the DNx-DIO-480 counter/timer subsystem. See *Sample480CT.c* for example code.

Table 5-3 Low-level Counter API

Function	Description
`DqAdv480CTSetInputSource`	Set DIO-480 counter/timer input sources.
`DqAdvCTStartCounter`	Start or stop counter channel
`DqAdvCTClearCounter`	Reset counter to the initial value in the load register.
`DqAdvCTRead`	Read data from a counter.
`DqAdvCTWrite`	Change CLKOUT signal by writing to CR0 and CR1.
`DqAdvCTCfgCounter`	Configure advanced counter settings.
`DqAdvCTCfgForGeneralCounting`	Configure counter as a general event counter or timer.
`DqAdvCTCfgForBinCounter`	Configure counter to count the number of events in a specific time interval.
`DqAdvCTCfgForPeriodMeasurement`	Configure counter to measure how long CLKIN is high and how long CLKIN is low over N periods.
`DqAdvCTCfgForHalfPeriod`	Configure counter to measure pulse width of CLKIN.
`DqAdvCTCfgForTPPM`	Configure counter to measure the average period of CLKIN over the user-defined time interval.
`DqAdvCTCfgForQuadrature`	Configure counter as a quadrature decoder; GATE pin defines direction of counting.
`DqAdvCTCfgForPWM`	Configure counter for PWM output.
`DqAdvCTCfgForPWMTrain`	Configure counter to output a set number of PWM pulses.

5.4.2.1 Configuration Settings

Each counter can be independently configured using either `DqAdvCTConfigCounter()` or one of the `DqAdvCTCfg___()` functions. `DqAdvCTConfigCounter()` is the lowest level configuration function. However, since not all parameter combinations are supported in all modes, it is easier to use a `DqAdvCTCfg___()` function when possible. `DqAdvCTCfg___()` automatically selects the best counting mode for the application and only exposes relevant parameters.

Parameter	Description
`startmode`	Auto-start or start on `DqAdvCTStartCounter()`
`sampwidth`	PWM sample width
`ps`	Prescaler value for clock division
`pc`	Period count register; used when measuring multiple periods
`cr0`	Compare register 0, CLKOUT is low between `lr` and `cr0`
`cr1`	Compare register 1, CLKOUT is high between `cr0` and `cr1`
`tbr`	Timebase register; used for timed measurements
`dbg`	Input debouncing gate register; GATE to be stable

Parameter	Description
`dbc`	Input debouncing clock register; defines time for CLKIN to be stable
`iie`	Invert CLKIN pin
`gie`	Invert GATE pin
`oie`	Invert CLKOUT pin
`mode`	Counting mode (Section 5.4.2.2)
`trs`	Use GATE as trigger
`enc`	Auto-clear counter after end_mode and await next trigger
`gated`	Use GATE to enable/disable counter, if GATE is not already being used as a trigger
`re`	Restart counter after end_mode condition is met
`end_mode`	Count termination condition (Section 5.4.2.3)
`lr`	Load register; sets initial value of the counter

5.4.2.2 Counting Modes

The following modes are selectable in `DqAdvCTConfigCounter()`:

- **Basic Timer** - counts the number of 66 MHz clock cycles (or cycles of 66 MHz divided by the prescaler). The output stays low as the counter counts from `lr` up to `cr0` and then stays high until it reaches `cr1`. The counter may be used as a Bin Counter or generate a One-Shot Output by selecting an appropriate end mode (Section 5.4.2.3).
- **External Event Counter** - similar to the Basic Timer, except the clock source is the debounced CLKIN signal rather than the 66 MHz clock.
- **Timed Pulse Period Measurement** - counts the total number of rising CLKIN edges over the `tbr` time interval, as well as the total number of 66 MHz clock cycles between the first and last rising edge. The average period can be computed from these two measurements.
- **Half-period Capture** - counts the number of 66 MHz clock cycles over which CLKIN is high. The pulse width can then be calculated.
- **N-period Capture** - counts the number of 66 MHz clock cycles for both the positive and negative parts of CLKIN until `pc`-1 number of periods have elapsed. The average period can then be calculated.
- **Quadrature Decoder** - counts the number of rising CLKIN edges, counting up if GATE=1 and down if GATE=0.

All modes except Quadrature Decoder support an optional hardware trigger.

5.4.2.3 End Modes

The following count termination conditions are available:

- Count register reaches CR0 value
- Count register reaches CR1 value
- Count register reaches 0xFFFFFFFF
- Period count register reaches 0
- Timebase register reaches 0
- GATE goes from high to low

5.5 Async Events API

Most asynchronous event-handling functions are board-agnostic and described in the *“PowerDNA API Reference Manual”*. There is only one function specific to the DNx-DIO-480. Please see *SampleAsync480* for an example of how to configure and retrieve event packets.

Table 5-4 Low-level Asynchronous Events API

	Function	Description
Async	`DqAdv480ConfigEvents`	Configure the board to send status data upon one of the following events: • DIO pin changes state • Periodically at a user-defined rate

5.6 RtDMap API

Real Time Data Map (RtDMap) mode uses the same API as Point-by-Point mode for channel configuration (Section 5.4); however, generic DMap functions are used for reading data. DMap API is documented in the *“PowerDNA API Reference Manual*”.

Refer to *SampleRTDMap480* for an example of how to set up a Data Map on the DNx-DIO-480. The following DNx-DIO-480 channels can be added to the DMap:

Table 5-5 DMap Channels

Subsystem	Channels	Notes
`DQ_SS0IN`	`DQ_DIO480_DMAP_DIO_IN`	Read industrial DIO current state
	`DQ_DIO480_DMAP_DIO_IN_DEB`	Read industrial DIO debounced state
`DQ_SS0OUT`	`DQ_DIO480_DMAP_DIO_OUT`	Write industrial DIO state
`DQ_SS1IN`	`DQ_DIO480_DMAP_TTL_IN`	Read TTL current state
`DQ_SS1OUT`	`DQ_DIO480_DMAP_TTL_OUT`	Write TTL state

A basic overview of DMap usage is provided in Section 5.6.1. More information on RtDMap is available in the *“PowerDNx Protocol Manual”*.

5.6.1 DMap Tutorial

As shown in *SampleRtDMap480*, a DMap program is structured as follows:

DMap Configuration:

1. Create a DMap.
2. Configure input channels.
3. Add input channels to the DMap.
4. Configure ouput channels.
5. Add output channels to the DMap.
6. Start the DMap.

DMap Operation:

7. Schedule output data to write upon next refresh.
8. Refresh the DMap.
9. Read retrieved data from input channels (returned in reply to refresh).

Close Out DMap:

10. Stop and close the DMap.

5.6.1.1 DMap Configuration

1. To create a new DMap, call `DqRtDmapInit()`. One copy of the DMap is stored on the IOM and another is stored on the host. During operation (Step 8), the IOM will update its version of the map.

```
//Create and initialize a DMap. Set refresh rate to 0.0.

DqRtDmapInit(hd, &dmapid, 0.0);
```

2. Configure input channels using Point-by-Point API. This step will focus on industrial digital input; additional subsystems are covered in the example code.

```
//Set up an input channel list.

for(i=0; i<32; i++){
   configure_cl[i] = i;
   low_level[i] = INPUT_LEVEL_LOW;
   high_level[i] = INPUT_LEVEL_HIGH;
   debounce_samples[i] = INPUT_DEBOUNCE_SAMPLES;
}

//Configure 32 industrial input channels.

DqAdv480DIOSetLevels(hd, devn, 32, configure_cl, low_level, high_level);
DqAdv480DIOSetDebouncer(hd, devn, 32, configure_cl, debounce_samples);
```

3. Add input channels to the DMap with their corresponding subsystem names (Table 5-6).

```
// Add industrial DIO input state channel to the DMap.

dmap_chan_entry = DQ_DIO480_DMAP_DIO_IN;
DqRtDmapAddChannel(hd, dmapid, devn, DQ_SS0IN, &dmap_chan_entry, 1);

// Add industrial DIO input debounced state channel to the DMap.

dmap_chan_entry = DQ_DIO480_DMAP_DIO_IN_DEB;
DqRtDmapAddChannel(hd, dmapid, devn, DQ_SS0IN, &dmap_chan_entry, 1);
```

4. Configure output channels using Point-by-Point API. This step will focus on industrial digital output; additional subsystems are covered in the example code.

```
//Set up an output channel list.

for(i=0; i<32; i++){
   configure_cl[i] = i;
   output_mode[i] = DQ_DIO480_DIO_MODE_PUSH_PULL;
   // Use DQ_DIO480_DIO_SRC_DOUT for direct write control
   output_source[i] = DQ_DIO480_DIO_SRC_DOUT;
}

//Configure the output mode and source for 32 industrial output channels.

DqAdv480DIOSetOutputMode(hd, devn, configure_cl_size, configure_cl,
         output_mode);
DqAdv480DIOSetSource(hd, devn, configure_cl_size, configure_cl,
         output_source);
```

5. Add the channels to the DMap with their corresponding subsystem names (Table 5-6).

```
// Add industrial DIO output state channel to the DMap.

dmap_chan_entry = DQ_DIO480_DMAP_DIO_OUT;
DqRtDmapAddChannel(hd, dmapid, devn, DQ_SS0OUT, &dmap_chan_entry, 1);
```

6. Start the DMap with the configuration and channels requested above.

```
//Start the DMap.

DqRtDmapStart(hd, dmapid);
```

5.6.1.2 DMap Operation

7. `DqRtDmapWriteRawData32()` writes output channel values to the host map. The DMap can hold one data point per channel. However, data is not actually transferred to the IOM until the `DqRtDmapRefresh()` call in Step 8.

```
//Set bits 0-3 high on digital output, the other bits will be low

uint32 write_buffer[] = 0x0f;
int write_data_size = 1;
DqRtDmapWriteRawData32(hd, dmapid, devn, write_buffer, write_data_size);
```

8. Calling `DqRtDmapRefresh()` sends the output data from the host to the IOM. On the reply, the IOM transfers one data point per configured input channel to the host.

```
//Send output data and receive input data.

DqRtDmapRefresh(hd, dmapid);
```

9. Input data can be read from the host's version of the map using `DqRtDmapReadRawData32()` or `DqRtDmapReadScaledData()`.

```
//Read data from the latest DqRtDmapRefresh.
//Set dmap_cl_in_size equal to the number of input channels configured

DqRtDmapReadRawData32(hd, dmapid, devn, read_buffer, dmap_cl_in_size);
```

5.6.1.3 Close Out DMap

10. Stop and clean up the DMap with the calls:

```
DqRtDmapStop(hd, dmapid);

DqRtDmapClose(hd, dmapid);
```

5.7 RtVMap API

VMap uses the same API as Point-by-Point mode for channel configuration (Section 5.4); however, generic VMap functions are used for reading data. The VMap API is documented in the *“PowerDNA API Reference Manual.”*

Refer to *SampleVMap480* for an example of how to set up and run a Variable Map (VMap) on the DNx-DIO-480. Table 5-6 lists all of the DNx-DIO-480 channels that can be added to the VMap.

Table 5-6 VMap Channels

Subsystem	Channels	Notes
`DQ_SS0IN`	DQ_DIO480_VMAP_DIO_IN	Debounced DIO state data then time stamp is added to the FIFO. This results in a constant data stream of data and time stamp, regardless of the input state.
	DQ_DIO480_VMAP_DIO_IN_COS	Debounced DIO state data then time stamp is added to the FIFO only after a change-of-state occurs.

A basic overview of VMap usage is provided in Section 5.7.1. More detailed information on RtVMap can be found in the *“PowerDNx Protocol Manual”*.

5.7.1 VMap Tutorial

As shown in *SampleVMap480*, a VMap program is structured as follows:

VMap Configuration:

1. Setup the VMap channel list
2. Set up per channel configuration.
3. Create a VMap.
4. Add channels to the VMap
5. Start the VMap.

VMap Operation (loop):

6. Set up request for data
7. Refresh the VMap.
8. Read data from the last refresh.
9. Iterate through each sample of a scan

Close Out VMap:

10. Stop and close the VMap.

5.7.1.1 VMap Configuration

1. Declare and set-up the VMAP channel list. Because the DIO-480 always returns the state of all 32 channels as one 32-bit word, the VMAP will be set up as if there is only one channel. Note that time stamp will always be included with the input data..

```
//Setup VMap request size

uint32 vmap_samples_per_scan = 2;   // 1 input data sample + 1 time stamp
uint32 vmap_scan_size = vmap_samples_per_scan * sizeof(uint32);
uint32 vmap_rq_size = VMAP_MAX_IN_SCANS * vmap_scan_size;

// VMap input data buffer. The size is equal to
// VMAP_MAX_IN_SCANS * number of data points in a scan.
// In the DIO-480 case, each scan is the input state + time stamp.

uint32 vmap_in_bdata[VMAP_MAX_IN_SCANS * 2];
uint32 vmap_bdata_sample;
```

2. Set up the per channel configuration.

```
for (i = 0; i < configure_cl_size; i++) {
    configure_cl[i] = i;

    // Set all channels to tri-state for input
    output_mode[i] = DQ_DIO480_DIO_MODE_FLOAT;
    output_source[i] = DQ_DIO480_DIO_SRC_DOUT;

    // Set low and high voltage levels
    low_level[i] = INPUT_LEVEL_LOW;
    high_level[i] = INPUT_LEVEL_HIGH;

    // Set input debouncer
    debounce_samples[i] = INPUT_DEBOUNCER;
}

DqAdv480DIOSetOutputMode(hd, DEVN, configure_cl_size, configure_cl,
     output_mode);
DqAdv480DIOSetSource(hd, DEVN, configure_cl_size, configure_cl,
     output_source);
DqAdv480DIOSetLevels(hd, DEVN, configure_cl_size, configure_cl,
     low_level, high_level);
DqAdv480DIOSetDebouncer(hd, DEVN, configure_cl_size, configure_cl,
     debounce_samples);
```

3. To create a new VMap, call `DqRtVmapInit()`. One copy of the VMap is stored on the IOM and another is stored on the host. During operation (Step 7), the IOM will update its version of the map at the rate specified during initialization.

```
//Create and initialize a VMap with a 5000 Hz refresh rate.

DqRtVmapInit(hd, &vmapid, 5000);
```

4. Add the channels to the VMap with their corresponding subsystem names (Table 5-6). The `SetChannelList()` function identifies the number of physical channels on the DNx-DIO-480.

```
//Create channel list for VMap. Selecting DQ_DIO480_VMAP_DIO_IN_COS
//will result in data being added to the FIFO only on a change-of-state.
//Selecting DQ_DIO480_VMAP_DIO_IN will result in data being added to the
//FIFO at the specified VMAP rate.

vmap_cl[0] = DQ_DIO480_VMAP_DIO_IN_COS;

//Create flags for the channel list.

vmap_cl_flags[0] = DQ_VMAP_FIFO_STATUS;

// Add the channel to the VMap, and configure the input rate

DqRtVmapAddChannel(hd, vmapid, DEVN, DQ_SS0IN, vmap_cl, vmap_cl_flags,
     vmap_cl_size);
DqRtVmapSetScanRate(hd, vmapid, DEVN, DQ_SS0IN, VMAP_RATE);
```

5. Start the VMap with the configuration and channels requested above.

```
//Start the VMap.

DqRtVmapStart(hd, vmapid);
```

5.7.1.2 VMap Operation

Loop through the steps in VMap Operation.

6. Read data out of the VMAP. Note that data is not actually transferred to the IOM until the `DqRtVmapRefresh()` call.

```
// Setup request for data that will occur on next DqRtVmapRefresh call

DqRtVmapRqInputDataSz(hd, vmapid, 0, vmap_rq_size, &vmap_act_size, NULL);
```

7. The VMap request has been prepared, so the command can be sent with `DqRtVmapRefresh()..`

```
//Send output data and receive input data.

DqRtVmapRefresh(hd, vmapid, 0);
```

8. Get data from the last refresh.

```
// Get data from the last DqRtVmapRefresh call

DqRtVmapGetInputData(hd, vmapid, 0, vmap_rq_size, &vmap_data_size,
&vmap_avl_size, (uint8*)vmap_in_bdata);
```

9. Iterate through each received sample of each scan.

```
// Iterate through each received sample of each scan

for (scan_idx = 0; scan_idx < vmap_cur_scans_rcvd; scan_idx++) {
        for (sample_idx = 0; sample_idx < (int)vmap_samples_per_scan;
             sample_idx++) {
               vmap_bdata_sample = DqNtohl(hd, vmap_in_bdata[(scan_idx *
                         vmap_samples_per_scan) + sample_idx]);
                if (sample_idx == 0) {
                        // The first sample of a scan is input state
                        input_state = vmap_bdata_sample;
                } else {
                        // The second sample of a scan is time stamp
                       DqAdvRawToScaleValue(hd, DEVN, DQ_LNCL_TIMESTAMP,
                                 vmap_bdata_sample, &timestamp);
                      printf("State=0x%x\tTime stamp=%f\n", input_state,
                                 timestamp);
                }
        }
}
```

If a FIFO overflow error occurs, try reducing `IN_SCANRATE`, increasing `OUT_SCANRATE`, or increasing the `DqRtVmapRefresh()` rate.

5.7.1.3 Close Out VMap

10. Stop and clean up the VMap with the calls:

```
DqRtVmapStop(hd, vmapid);

DqRtVmapClose(hd, vmapid);
```

5.8 AVMap API

AVMap uses the same API as Point-by-Point mode for channel configuration (Section 5.4); however, generic AVMap functions are used for reading data.

Refer to *SampleAVMap480* for an example of how to set up and run an Asynchronous Variable Map (AVMap) on the DNx-DIO-480. The example program also provides more detail on declaring and initializing the variables used in the following tutorial. Table 5-7 lists the DNx-DIO-480 channels that can be added to the AVMap.

Table 5-7 AVMap Channels

Subsystem	Channels	Notes
`DQ_SS0IN`	DQ_DIO480_VMAP_DIO_IN	Debounced DIO state data then time stamp is added to the FIFO. This results in a constant data stream of data and time stamp, regardless of the input state.
	DQ_DIO480_VMAP_DIO_IN_COS	Debounced DIO state data then time stamp is added to the FIFO only after a change-of-state occurs.

5.8.1 AVMap Tutorial

This section provides a basic overview of AVMap usage. As shown in *SampleAVMap480*, an AVMap program is structured as follows:

AVMap Configuration:

1. Create a VMap.
2. Configure input/output channels.
3. Add input/output channels to the VMap.
4. Configure DNx-DIO-480 scan rates.
5. Start the VMap.

AVMap Operation:

6. Schedule input data to read upon next refresh.
7. Refresh the VMap and get data

Close Out AVMap:

8. Stop and close the VMap.

5.8.1.1 AVMap Configuration

1. Declare and set-up the VMAP channel list. Because the DIO-480 always returns the state of all 32 channels as one 32-bit word, the VMAP will be set up as if there is only one channel. Note that time stamp will always be included with the input data..

```
//Setup AVMap transfer size

uint32 vmap_samples_per_scan = 2;   // 1 input data sample + 1 time stamp
uint32 vmap_scan_size = vmap_samples_per_scan * sizeof(uint32);
uint32 vmap_rq_size = AVMAP_SAMPLES * sizeof(uint32);

//AVMap input data buffer. The size is dependent on AVMAP_MODE selected.

uint32 vmap_in_bdata[AVMAP_SAMPLES];
uint32 vmap_bdata_sample;
```

2. Set up the per channel configuration.

```
//Set up output mode, input levels, and debouncer for each channel

for (i = 0; i < configure_cl_size; i++) {
    configure_cl[i] = i;
    // Set all channels to known state, tri-state for input
    output_mode[i] = DQ_DIO480_DIO_MODE_FLOAT;
    output_source[i] = DQ_DIO480_DIO_SRC_DOUT;
    // Set channels input levels and debouncer
    low_level[i] = INPUT_LEVEL_LOW;
    high_level[i] = INPUT_LEVEL_HIGH;
    // Set debounce samples
    debounce_samples[i] = INPUT_DEBOUNCER;
}

//Call configuration functions

DqAdv480DIOSetOutputMode(hd, DEVN, configure_cl_size, configure_cl,
     output_mode);
DqAdv480DIOSetSource(hd, DEVN, configure_cl_size, configure_cl,
     output_source);
DqAdv480DIOSetLevels(hd, DEVN, configure_cl_size, configure_cl,
     low_level, high_level);
DqAdv480DIOSetDebouncer(hd, DEVN, configure_cl_size, configure_cl,
     debounce_samples);
```

3. To create a new AVMap, call `DqRtVmapInit()`..

```
//Create the VMap

DqRtVmapInit(hd, &vmapid, VMAP_RATE);
```

4. Add the channels to the VMap with their corresponding subsystem names (Table 5-7). The `SetChannelList()` function identifies the number of physical channels on the DNx-DIO-480.

```
//Create channel list for VMap. Selecting DQ_DIO480_VMAP_DIO_IN_COS
//will result in data being added to the FIFO only on a change-of-state.
//Selecting DQ_DIO480_VMAP_DIO_IN will result in data being added to the
//FIFO at the specified VMAP rate.

vmap_cl[0] = DQ_DIO480_VMAP_DIO_IN_COS;

//Create flags for the channel list.

vmap_cl_flags[0] = DQ_VMAP_FIFO_STATUS;

// Add the channel to the VMap, and configure the input rate

DqRtVmapAddChannel(hd, vmapid, DEVN, DQ_SS0IN, vmap_cl, vmap_cl_flags,
     vmap_cl_size);
DqRtVmapSetScanRate(hd, vmapid, DEVN, DQ_SS0IN, VMAP_RATE);
```

5. Setup AVMap mode.

```
//Start the VMap.

DqRtAXMapStart(hd, vmapid, AVMAP_MODE, AVMAP_TIME_RATE,
     AVMAP_WMRK_SAMPLES, 0);
```

5.8.1.2 AVMap Operation

6. Read data out of the VMAP. Note that data is not actually transferred to the IOM until the `DqRtVmapRefresh()` call.

```
// Setup request for data that will occur on next DqRtVmapRefresh call

DqRtAXMapSlotAllocate(hd, TRUE, vmapid, 0);
DqRtVmapRqInputDataSz(hd, vmapid, 0, vmap_rq_size, &vmap_act_size, NULL);

// Update data from the layer

DqRtVmapRefresh(hd, vmapid, 0);
DqRtAXMapEnable(hd, TRUE);
DqCmdSwTrigger(hd, (1L << DEVN));
```

7. *Loop through the remaining steps in AVMap Operation*.

```
//Refresh Inputs

DqRtAVmapRefreshInputsExt(hd, vmapid, &pkttype, &counter, &pkttimestamp, NULL)

// Get data from the last DqRtVmapRefresh call

DqRtVmapGetInputData(hd, vmapid, 0, vmap_rq_size, &vmap_data_size,
     &vmap_avl_size, (uint8*)vmap_in_bdata);

// Iterate through each received sample of each scan

vmap_cur_scans_rcvd = vmap_data_size / vmap_scan_size;
vmap_scans_rcvd += vmap_cur_scans_rcvd;
for (scan_idx = 0; scan_idx < vmap_cur_scans_rcvd; scan_idx++) {
     for (sample_idx = 0; sample_idx < (int)vmap_samples_per_scan;
          sample_idx++) {
            vmap_bdata_sample = DqNtohl(hd, vmap_in_bdata[(scan_idx *
                      vmap_samples_per_scan) + sample_idx]);
            if (sample_idx == 0) {
                 // The first sample of a scan is input state
                 input_state = vmap_bdata_sample;
            } else {
                 // The second sample of a scan is time stamp
                 DqAdvRawToScaleValue(hd, DEVN, DQ_LNCL_TIMESTAMP,
                             vmap_bdata_sample, &timestamp);
                 printf("State=0x%x\tTime stamp=%f\n", input_state,
                             timestamp);
            }
     }
}
```

If a FIFO overflow error occurs, try reducing `IN_SCANRATE`, increasing `OUT_SCANRATE`, or increasing the `DqRtVmapRefresh()` rate.

5.8.1.3 Close Out AVMap

8. Stop and clean up the AVMap with the calls:

```
DqRtAXMapEnable(hd, FALSE);
DqRtVmapStop(hd, vmapid);
DqRtVmapClose(hd, vmapid);
```

5.9 H-Bridge Configuration

Industrial digital output lines can be configured as H-Bridge pairs. This allows for implementation of the four standard DC motor states (drive right, drive left, brake, and coast). The low-level API provides the function listed in Table 5-8 for creating an H-Bridge configuration for DNx-DIO-480 industrial digital output pairs.

Table 5-8 H-Bridge Configuration API

Function	Description
`DqAdv480DIOSetHBridge`	Set DIO-480 industrial DIO H-Bridge configuration.

For H-Bridge configuration, first follow the same steps used for configuring digital output. H-Bridge configuration links two adjacent channels together so that the different H-Bridge states can be easily implemented. Without this functionality, the user application would need to frequently change output modes and perform write operations to achieve the same behavior. Use the following function call and channel pair definitions to configure the H-Bridge pairs. You only need to use the channel pairs required by your application.

```
DqAdv480DIOSetHBridge(hd, devn,
    DQ_DIO480_H_BRIDGE_0_1   |  // Enable H-Bridge channel pair 0 and 1
    DQ_DIO480_H_BRIDGE_2_3   |  // Enable H-Bridge channel pair 2 and 3
    DQ_DIO480_H_BRIDGE_4_5   |  // Enable H-Bridge channel pair 4 and 5
    DQ_DIO480_H_BRIDGE_6_7   |  // Enable H-Bridge channel pair 6 and 7
    DQ_DIO480_H_BRIDGE_8_9   |  // Enable H-Bridge channel pair 8 and 9
    DQ_DIO480_H_BRIDGE_10_11 |  // Enable H-Bridge channel pair 10 and 11
    DQ_DIO480_H_BRIDGE_12_13 |  // Enable H-Bridge channel pair 12 and 13
    DQ_DIO480_H_BRIDGE_14_15 |  // Enable H-Bridge channel pair 14 and 15
    DQ_DIO480_H_BRIDGE_16_17 |  // Enable H-Bridge channel pair 16 and 17
    DQ_DIO480_H_BRIDGE_18_19 |  // Enable H-Bridge channel pair 18and 19
    DQ_DIO480_H_BRIDGE_20_21 |  // Enable H-Bridge channel pair 20 and 21
    DQ_DIO480_H_BRIDGE_22_23 |  // Enable H-Bridge channel pair 22 and 23
    DQ_DIO480_H_BRIDGE_24_25 |  // Enable H-Bridge channel pair 24 and 25
    DQ_DIO480_H_BRIDGE_26_27 |  // Enable H-Bridge channel pair 26 and 27
    DQ_DIO480_H_BRIDGE_28_29 |  // Enable H-Bridge channel pair 28 and 29
    DQ_DIO480_H_BRIDGE_30_31    // Enable H-Bridge channel pair 30 and 31);
```

The following H-Bridge motor states are provided:

```
DQ_DIO480_WRITE_H_BRIDGE_BRAKE
DQ_DIO480_WRITE_H_BRIDGE_L
DQ_DIO480_WRITE_H_BRIDGE_R
DQ_DIO480_WRITE_H_BRIDGE_COAST
```

When performing the write, shift the H-Bridge state to the even numbered channel of the pair. The following example writes the `DQ_DIO480_WRITE_H_BRIDGE_BRAKE` state to channels 10 and 11.

```
write_state = DQ_DIO480_WRITE_H_BRIDGE_BRAKE << 10;
DqAdv480DIOWrite(hd, devn, write_state);
```

5.10 Circuit Breaker Configuration

The industrial digital output lines are protected by circuit breakers. The low-level API provides the functions listed in Table 5-9 for configuring the circuit breakers and reading circuit breaker status.

Table 5-9 Circuit Breaker API

Function	Description
`DqAdv480CBSetEnable`	Enable or disable the circuit breakers on selected channels
`DqAdv480CBSetReadCount`	Set the number of consecutive out of limit reads that need to be made for circuit breakers to trip.
`DqAdv480CBReadStatus`	Read DIO-480 circuit breaker tripped status.
`DqAdv480CBReset`	Reset circuit breakers on selected channels of a DIO-480

Circuit breakers can be enabled for selected DO channels by providing a bitmask to `DqAdv480CBSetEnable()`. A 1 in a channel’s bit position enables limit checking for that channel.

```
//enable the circuit-breaker on all channels except 7 and 8.

uint32 enable_mask = 0xfffffe7f;
DqAdv480CBSetEnable(hd, devn, enable_mask);
```

By default, the circuit breakers won’t trip until 1500 consecutive out of range samples occur. The number of out of range samples required to trip the circuit breakers is configurable by calling `DqAdv480CBSetReadCount()`. The minimum number of samples that can be specified must fall within the range of `DQ_DIO480_CB_COUNT_MIN` and `DQ_DIO480_CB_COUNT_MAX`.

```
//set the number of consecutive samples needed to trip the circuit
//breakers to the minimum count

DqAdv480CBSetReadCount(hd, devn, DQ_DIO480_CB_COUNT_MIN);
```

The tripped status of the circuit breakers can be obtained by calling `DqAdv480CBReadStatus()`. A 1 in a channel’s bit position means its circuit breaker has tripped.

```
//obtain the tripped state.

uint32 tripped_state = 0;
DqAdv480CBReadStatus(hd, devn, &tripped_state);
```

Circuit breakers on selected DO channels can be reset by providing a bitmask to `DqAdv480CBReset()`. A 1 in a channel’s bit position will reset the circuit channel for that channel.

```
//reset the circuit-breaker for all channels except 7 and 8.

uint32 reset_mask = 0xfffffe7f;
DqAdv480CBReset(hd, devn, reset_mask);
```

5.11 Guardian Diagnostics

The DNx-DIO-480 is part UEI's Guardian Series and provides real-time diagnostic information. The low-level API provides the functions listed in Table 5-10 for reading diagnostic information from the DNx-DIO-480.

Table 5-10 Low-level Guardian Diagnostic API

Function	Description
`DqAdv480DIOReadChannelVoltage`	Read voltage on industrial DIO lines.
`DqAdv480DIOReadVccVoltage`	Read diagnostic DIO-480 Vcc ADC values of the channel groups
`DqAdv480TemperatureRead`	Read DIO-480 temperature

ADC voltage values for each of the industrial DIO channels can be obtained by providing a channel list to `DqAdv480DIOReadChannelVoltage()`. Calibrated binary data and/or converted voltage data is returned in buffers provided to the function.

```
//read channel voltage
//initialize ch_list with channel numbers

int ch_list_size = DQ_DIO480_DIO_CHAN;  //size of channel list
uint32 ch_list[DQ_DIO480_DIO_CHAN + 1]; //channel list + 1 for time stamp
uint32 bdata[DQ_DIO480_DIO_CHAN + 1];   //calibrated binary data + 1
double fdata[DQ_DIO480_DIO_CHAN + 1];   //converted voltage data + 1

DqAdv480DIOReadChannelVoltage(hd, devn,
              ch_list_size, ch_list, bdata, fdata);
```

Vcc supply voltage values for each of the eight Vcc groups can be obtained by providing a channel list to `DqAdv480DIOReadVccVoltage()`. Calibrated binary data and/or converted voltage data is returned in buffers provided to the function.

```
//read Vcc voltage
//initialize ch_list with channel numbers

int ch_list_size = 8;                   //number of Vcc groups
uint32 ch_list[DQ_DIO480_DIO_CHAN + 1]; //channel list + 1 for time stamp
uint32 bdata[DQ_DIO480_DIO_CHAN + 1];   //calibrated binary data + 1
double fdata[DQ_DIO480_DIO_CHAN + 1];   //converted voltage data + 1

DqAdv480DIOReadVccVoltage(hd, devn,
              ch_list_size, ch_list, bdata, fdata);
```

The temperature of the DNx-DIO-480 can be obtained by calling `DqAdv480TemperatureRead()`.

```
//read DIO-480 temperature

uint32 temp_raw;      //Raw temperature sensor data
double temp_scaled;   //converted temperature sensor data
DqAdv480TemperatureRead(hd, devn, &temp_raw, &temp_scaled);
```

Appendix A

Accessories

A.1 General Purpose STP Board and Cable

The DNx-DIO-480 is compatible with UEI's general purpose 62-pin cable and screw terminal board. This may be an attractive alternative when space is at a premium or your application is not switching high frequency or high power digital signals.

DNA-CBL-62

The DNA-CBL-62 is a 62-way, round, heavy shielded cable with 62-pin male D-sub connectors on both ends. The cable is 2.5 ft (75 cm) long, weighs 9.49 ounces or 269 grams.

The cable is also available in the following lengths:

- 10 ft (3.05 m) P/N DNA-CBL-62-10
- 20 ft (6.10 m) P/N DNA-CBL-62-20
- 40 ft (12.20 m) P/N DNA-CBL-62-40

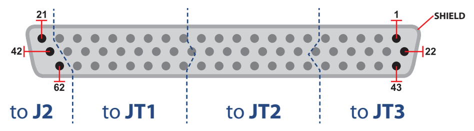
DNA-STP-62

The STP-62 is a Screw Terminal Panel with three 20-position terminal blocks (JT1, JT2, and JT3) plus one 3-position terminal block (J2). The dimensions of the STP-62 board are 4w x 3.8d x 1.2h inch (10.2 x 9.7 x 3 cm) (with standoffs). The weight of the STP-62 board is 3.89 ounces (110 grams).

DB-62 (female)
62-pin connector:

62	to J2	42	to J2	21	to J2
61	to JT1	41	to JT1	20	to JT1
60	to JT1	40	to JT1	19	to JT1
59	to JT1	39	to JT1	18	to JT1
58	to JT1	38	to JT1	17	to JT1
57	to JT1	37	to JT1	16	to JT1
56	to JT1	36	to JT1	15	to JT1
55	to JT2	35	to JT1	14	to JT1
54	to JT2	34	to JT2	13	to JT2
53	to JT2	33	to JT2	12	to JT2
52	to JT2	32	to JT2	11	to JT2
51	to JT2	31	to JT2	10	to JT2
50	to JT2	30	to JT2	9	to JT2
49	to JT2	29	to JT2	8	to JT2
48	to JT3	28	to JT3	7	to JT2
47	to JT3	27	to JT3	6	to JT3
46	to JT3	26	to JT3	5	to JT3
45	to JT3	25	to JT3	4	to JT3
44	to JT3	24	to JT3	3	to JT3
43	to JT3	23	to JT3	2	to JT3
		22	to JT3	1	to JT3

JT3 — 20-position **terminal block**:	**JT2** — 20-position **terminal block**:	**JT1** — 20-position **terminal block**:	**J2** — 5-position **terminal block**:	
22	55	20	5	SHIELD
1	13	41	4	GND
43	34	61	3	62
23	54	19	2	42
2	12	40	1	21
44	33	60		
24	53	18		
3	11	39		
45	32	59		
25	52	17		
4	10	38		
46	31	58		
26	51	16		
5	9	37		
47	30	57		
27	50	15		
6	8	36		
48	29	56		
28	49	14		
GND	7	35		



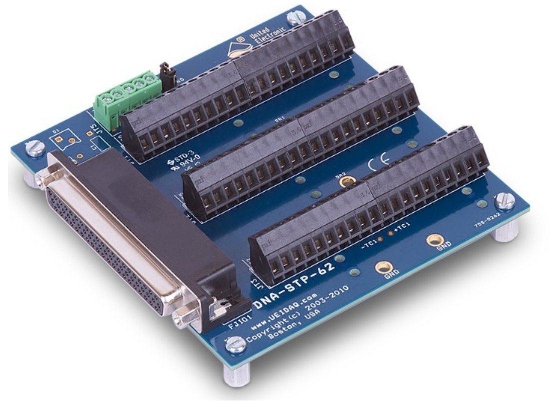


Figure A-1 Pinout and Photo of DNA-STP-62 Screw Terminal Panel