United Electronic Industries

The High-Performance Alternative

DNx-RTD-388

—

User Manual

8-Channel, fully isolated resistance simulator board for the PowerDNA Cube and RACK series chassis

July 2019

PN Man-DNx-RTD-388

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

Contacting United Electronic Industries

Mailing Address:

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

For a list of our distributors and partners in the US and around the world, please 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: support@ueidaq.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

Chapter 1 Introduction

This document outlines the feature set and use of the DNx-RTD-388 Resistance Temperature Detector (RTD) simulator boards.

The following product versions are described in this manual:

- DNx-RTD-388-1: simulates 1000 Ω RTD (180 Ω to 3900 Ω)
- DNx-RTD-388-100: simulates 100 Ω RTD (23 Ω to 390 Ω)

The following sections are provided in this chapter:

1.1 Organization of Manual

This *RTD-388 User Manual* is organized as follows:

- **Introduction**
  Chapter 1 provides an overview of DNx-RTD-388 RTD simulator board features, device architecture, connectivity, and logic.
- **Programming with the High-Level API**
  Chapter 2 provides an overview of the how to create a session, configure the session, and interpret results with the Framework API.
- **Programming with the Low-Level API**
  Chapter 3 is an overview of low-level API commands for configuring and using the RTD-388 series board.
- **Appendix A - Accessories**
  This appendix provides a list of accessories available for use with the DNx-RTD-388 board.
- **Index**
  This is an alphabetical listing of the topics covered in this manual.

NOTE: A glossary of terms used with the PowerDNA Cube/RACK and I/O boards can be viewed or downloaded from www.ueidaq.com.

Manual Conventions

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

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

NOTE: Notes alert you to important information.

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

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

Bold typeface will also represent field or button names, as in “Click **Scan Network**.”

Text formatted in `fixed` typeface generally represents source code or other text that should be entered verbatim into the source code, initialization, or other file.

Before you begin:

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.

No HOT SWAP

Always turn POWER OFF before performing maintenance on a UEI system. Failure to observe this warning may result in damage to the equipment and possible injury to personnel.

Usage of Terms

Throughout this manual, the term “Cube” refers to either a PowerDNA Cube product or to a PowerDNR RACKtangle^{TM} rack mounted system, whichever is applicable. The term DNR is a specific reference to the RACKtangle, DNA to the PowerDNA I/O Cube, and DNx to refer to both.

1.2 RTD-388 Board Overview

DNx-RTD-388 boards are 8-channel, high accuracy, fully isolated RTD simulators, compatible with UEI's Cube and RACK chassis.

DNA-RTD-388, DNR-RTD-388, and DNF-RTD-388 versions are designed for use with the UEI Cube, RACKtangle, and FLATRACK chassis respectively. These board versions are electronically identical except for the mounting hardware. The DNA version is designed to stack in a Cube chassis. The DNR/F versions are designed to plug into the backplane of a RACK chassis.

1.2.1 Output Design & Versions

Boards are based on actual switched resistors and precisely duplicate the behavior of the RTDs simulated.

The boards are available in two configurations. The DNx-RTD-388 series simulates a 1000 Ω RTD while the standard DNx-RTD-388-100 simulates the 100 Ω RTD. Other resistance values are available on a special order basis.

1.2.2 Applications

RTD-388 boards are an ideal solution for simulator / SIL applications where an on-board system device is expecting a RTD as an input. The boards are also an excellent solution for testing and diagnosing errors in a variety of RTD based systems.

1.2.3 Guardian Diagnostics

The DNx-RTD-388 series is part of UEI's Guardian series and provides powerful diagnostic readback functionality. A/D converters on each channel allow the application to monitor both input current and on-board runtime temperature. The board also provides simulation of open and short circuited RTDs.

1.2.4 Accessories

All connections are made through a convenient 37-pin D connector ensuring OEMs may easily obtain mating cables or connectors. Users may also connect the DNx-RTD-388 series boards to the DNA-STP-37 screw terminal panel via the DNA-CBL-37 series cables. The cables are fully shielded and are available in 3, 10 and 20 foot lengths.

1.2.5 Software Support

The DNx-RTD-388 series includes software drivers supporting all popular operating systems including Windows, Linux, QNX, VxWorks, and most other popular real-time operating systems. Windows users may take advantage of the powerful UEIDAQ Framework which provides a simple and complete software interface to all popular Windows programming language and data acquisition and control applications (e.g., LabVIEW, MATLAB).

1.3 Features

The RTD-388 board has the following features:

- 8 fully isolated channels
- Actual switched resistor configuration
- 1000 Ω (100 Ω and custom versions available)
- Guardian series diagnostics: input current and ADC on-die temperature readback
- Simulates both open and short circuited RTDs
- Wide ±4 mA excitation range
- Channel-to-channel and channel-to-chassis isolation

1.4 Specification

The technical specifications for the DNx-RTD-388-1 (1000 Ω) board are listed in **Table 1-1**.

The technical specifications for the DNx-RTD-388-100 (100 Ω) board are listed in **Table 1-2**.

Table 1-1 DNx-RTD-388-1 Technical Specifications (α=0.00385 unless otherwise noted)

Configuration	
Number of Channels	8
Maximum Excitation Current	± 4 mA
Resistance Specifications	
Nominal resistance	1000 Ω (0 °C)
Minimum resistance	180 Ω (-201.1 °C)
Maximum resistance	3900 Ω (849 °C)
Power-on resistance	Programmable. Default is 1000 Ω
Resolution (resistance)	Better than 0.5 Ω
Resolution (temp)	Better than 0.125°C (alpha = 0.00385)
Accuracy at ambient temp	
25 °C ±5 °C	0.26 °C (1.0 Ω)
-40 °C to +85 °C	1.0 °C (3.85 Ω)
Simulated Open RTD resistance	1 M Ω minimum
Simulated Shorted RTD resistance	1 Ω maximum
Resistance change update rate	0 - 100 Hz (This is how quickly the relays can switch)
Output voltage settling time	Dependent on excitation current and selected resistance
Output configuration	16.7 µF in parallel with selected output resistance
Diagnostic (Guardian) Read-back Specifications	
Input current range	± 5 mA
Current read-back resolution	± 10 µA
Current read-back accuracy	± 4 % of reading
Read-back update rate	up to 5 Hz
Temperature read-back accuracy	± 5 °C
General	
Power consumption	<3.0 W, not including IR dissipation
Operating range	Tested -40 to +85 °C
Humidity range	0-95%, non-condensing
Vibration *IEC 60068-2-6* *IEC 60068-2-64*	5 g, 10-500 Hz, sinusoidal 5 g (rms), 10-500Hz, broad-band 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
Altitude	to 70,000 feet
MTBF	Greater than 400,000 hours

Table 1-2 DNx-RTD-388-100 Technical Specifications (α=0.00385 unless otherwise noted)

Configuration	
Number of Channels	8
Maximum Excitation Current	± 4 mA
Resistance Specifications	
Nominal resistance	100 Ω (0 °C)
Minimum resistance	23 Ω (-189.5 °C)
Maximum resistance	390 Ω (849 °C)
Power-on resistance	Programmable. Default is 100 Ω
Resolution (resistance)	better than 0.05 Ω
Resolution (temp)	better than 0.125 °C (alpha = 0.00385)
Accuracy at ambient temp	
25 °C ±5 °C	1.0 °C (0.385 Ω)
-40 °C to +85 °C	4.0 °C (1.54 Ω)
Simulated Open RTD resistance	1 M Ω minimum
Simulated Shorted RTD resistance	1 Ω maximum
Resistance change update rate	0 - 100 Hz (This is how quickly the relays can switch)
Output voltage settling time	Dependent on excitation current and selected resistance
Output configuration	16.7 µF in parallel with selected output resistance
Diagnostic (Guardian) Read-Back Specifications	
Input current range	± 5 mA
Input current accuracy	± 100 µA
Board temp read-back accuracy	±5 °C
Read-back update rate	up to 5 Hz
General	
Power consumption	<3.0 W, not including IR dissipation
Operating range	Tested -40 to +85 °C
Humidity range	0-95%, non-condensing
Vibration *IEC 60068-2-6* *IEC 60068-2-64*	5 g, 10-500 Hz, sinusoidal 5 g (rms), 10-500Hz, broad-band 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
Altitude	to 70,000 feet
MTBF	greater than 400,000 hours

1.5 Device Architecture

Figure 1-1 shows a block diagram of the architecture of the RTD-388 board.

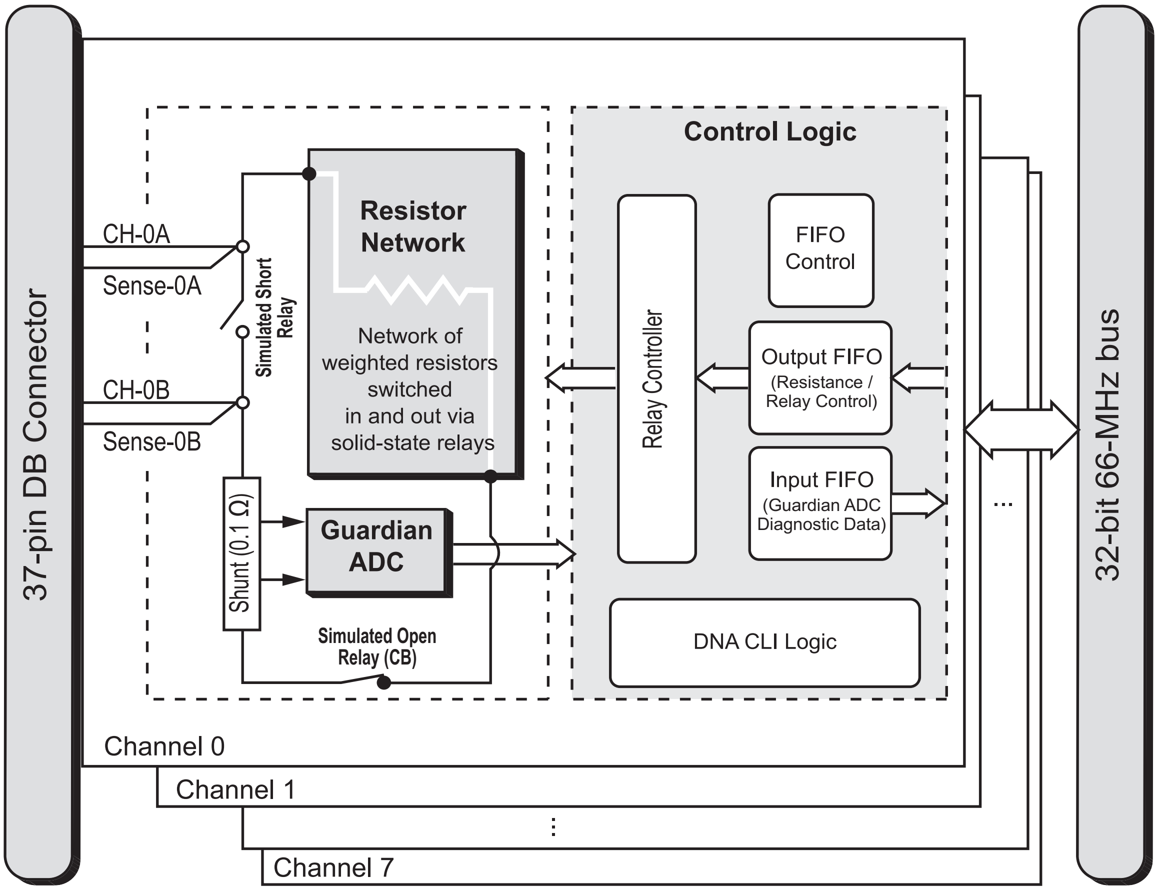

Figure 1-1 Block Diagram of the RTD-388 Board

1.5.1 Connector Pin Terminals & Sense Lines

Each RTD-388 board provides 8 isolated input channels designed for high-resolution RTD simulation.

Each channel provides an A, B, SenseA, and SenseB terminal through the DB-37 connector for connection to external circuitry.

The sense A/B lines are wired to the same nodes as their respective A/B terminals. Sense lines are provided as a convenience for wiring 3- and 4-wire RTD configurations.

The polarity of the A and B terminals is irrelevant in regards to the output resistance. However, if you are accessing the Guardian diagnostic features of the board, note that the Guardian readback circuitry assumes that A is the positive terminal and B is the negative terminal. If this polarity is reversed and the Guardian current is read, the provided readback value will be the opposite sign, (i.e., multiplied by -1). More information about Guardian circuitry is provided in “Guardian Diagnostic ADC & Relays” on page 8.

1.5.2 Resistor Network & Control

Each RTD-388 channel simulates RTD resistances using a network of switches and weighted resistors.

The RTD-388-1 accomplishes resistances from 180 Ω to 3900 Ω with a resolution of better than 0.5 Ω. The default resistance on power-up is 1000 Ω.

The RTD-388-100 accomplishes resistances from 23 Ω to 390 Ω. The default resistance on power-up is 100 Ω.

Control hardware connects resistors in series and/or parallel via solid-state relays based on user-programmed temperature or resistance settings. Using UEI API, users program resistance or temperature values, and the UEI library converts resistances in ohms to resistance data that encodes the required switch states, determining which resistors are switched in and switched out to provide the desired resistance.

Encoded resistance data is written to hardware, stored in a 256-word FIFO onboard the RTD-388 (see **Figure 1-1**), and accessed by board logic to set output resistance.

The output FIFO is accessible to standard DNx function calls and is written using software calls in either the UEIDAQ Framework (see **Chapter 2**) or the low-level API (see **Chapter 3**).

1.5.2.1 Hardware Delays for Resistance Settings

When programming a resistance, users can also program an associated delay for the control logic to wait before setting that resistance in hardware.

1.5.3 Guardian Diagnostic ADC & Relays

The RTD-388 Guardian diagnostic hardware includes per channel analog-to-digital converters for current and on-die ADC temperature monitoring, as well as relays for open and short circuit simulation.

Guardian diagnostics are user-programmable.

1.5.3.1 Current & Temperature Monitoring

Each RTD-388 channel is equipped with a 24-bit delta-sigma (ΔΣ) A/D converter, which is controlled by on-board logic and can be used to monitor current and on-die ADC temperature.

The Guardian ADC measures I_{in} current through the 0.1 Ω onboard shunt resistor, as well as provides temperature readings via an on-die temperature sensor (see **Figure 1-1**).

The sample rate of the Guardian ADC is 6.5 Hz.

Current and temperature measurements are stored in a 256-word FIFO that can be accessed via the UEIDAQ Framework (see **Chapter 2**) or the low-level API (see **Chapter 3**).

1.5.4 Open Circuit Simulation (Circuit Breaker)

Each channel includes an in-series relay that can be user-controlled for open circuit simulation. The relay also acts as an over-current and/or over-temperature circuit breaker, (refer to **Figure 1-1** for location).

By default, the relay is closed. In addition to users controlling the relay for open circuit simulation, control logic opens the circuit if Guardian current and temperature measurements exceed minimum and maximum limits.

The following are the default minimum and maximum values:

- Current limits are approximately ±4.5 mA
- ADC internal temperature limit is 105°C

You can reset the circuit breaker (close the relay) using the API for re-engaging the circuit breakers. Circuit breaker states can be read on a per channel basis via a status register.

1.5.5 Short Circuit Simulation

Each channel includes an additional relay located between Terminal A and Terminal B nodes that can be user-controlled for short circuit simulation (refer to **Figure 1-1** for location).

By default, the relay is open. When simulating a short circuit, users can open and close the short circuit relay with UEI API.

Optionally, users can set a maximum duration for the circuit to be shorted. If a duration is programmed, the channel will simulate a short circuit when the UEI short-circuit API executes, but the channel will open again automatically after the user-programmed duration.

The optional duration can be programmed in 100 µs increments, from 1 to 2^{32}-1.

1.5.6 Isolation

The control and data lines between the Control Logic and Resistor Network/ Guardian ADC are wired through an isolation barrier. The 350 V isolation barrier ensures that disturbances or damage to one channel does not affect other channels.

1.6 Indicators

The DNx-RTD-388 indicators are described in **Table 1-3** and illustrated in **Figure 1-2**.

Table 1-3 RTD-388 Indicators

LED Name	Description
RDY	Indicates board is powered up and operational
STS	Indicates which mode the board is running in: • **OFF**: Configuration mode, (e.g., configuring channels, running in point-by-point mode) • **ON**: Operation mode, (e.g., running in RtDMap mode)

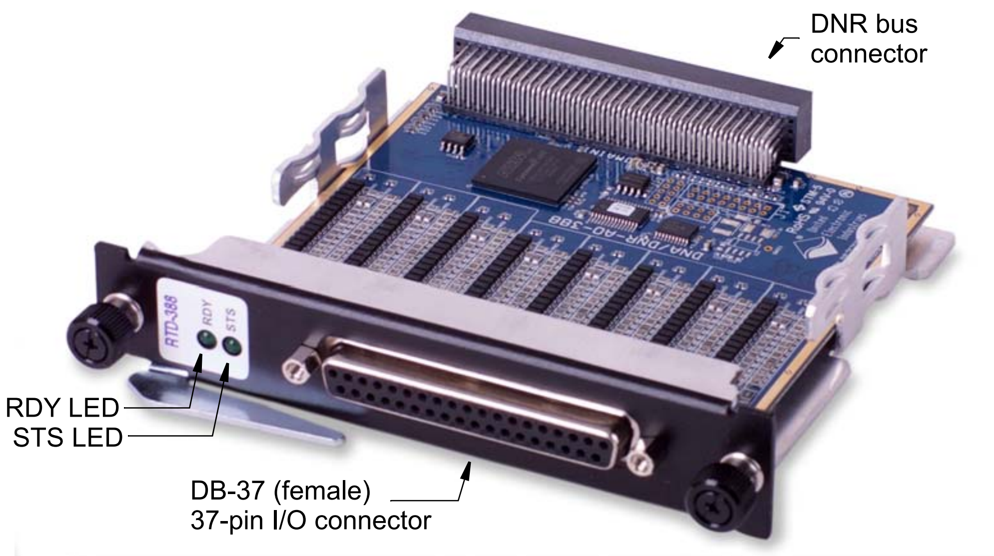

Figure 1-2 Photo of DNx-RTD-388 RTD Simulator Boards

1.7 Pinout

Figure 1-3 shows the pinout of the RTD-388. The RTD-388 board uses a 37-pin D-sub connector.

The following signals are located at the connector:

- CH-0A to CH-7A, Sense-0A to Sense-7A: A Terminal and Sense lines for channels 0 to 7
- CH-0B to CH-7B, Sense-0B to Sense-7B: B Terminal and Sense lines for channels 0 to 7
- nc: do not connect, Reserved

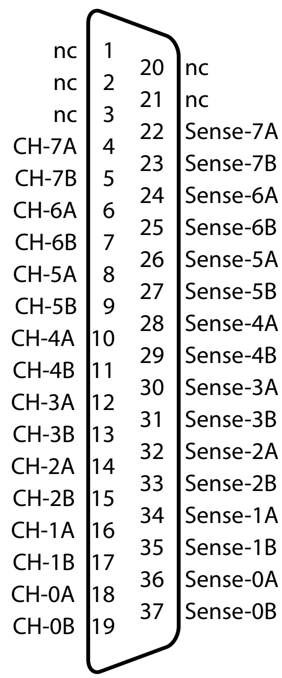

Figure 1-3 Pinout Diagram of the RTD-388 Board

Notes:

1. The output of the DNx-RTD-388 is resistance. The polarity of the A/B terminals is irrelevant in regards to the output resistance. However, the Guardian read-back circuitry assumes that “A” is the positive terminal and “B” is the negative. Customers are free to reverse this, but please note that the Guardian read-back provided will be the inverse polarity of the actual measurement (i.e. multiplied by -1).

2. The sense outputs are connected directly to the standard outputs. For example Sense-7A is connected to CH-7A on the DNx-RTD-388’s printed circuity board, and Sense-1B is connected to CH-1B (refer to **Figure 1-1**). The sense leads are provided as a convenience to those wiring the board in 3- and 4-wire modes.

1.8 PowerDNA Explorer for RTD-388

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 is provided in the installation directory.

When using PowerDNA Explorer to explore an RTD-388 board, note that the main panel contains the following tabs:

- **Diagnostic**: Displays diagnostic I_{in} and temperature readback values
- **Circuit Breaker**: Displays circuit breaker status and reengages
- **Output**: Sets output resistances immediately
- **Initialization**: Initializes output levels applied at power-up

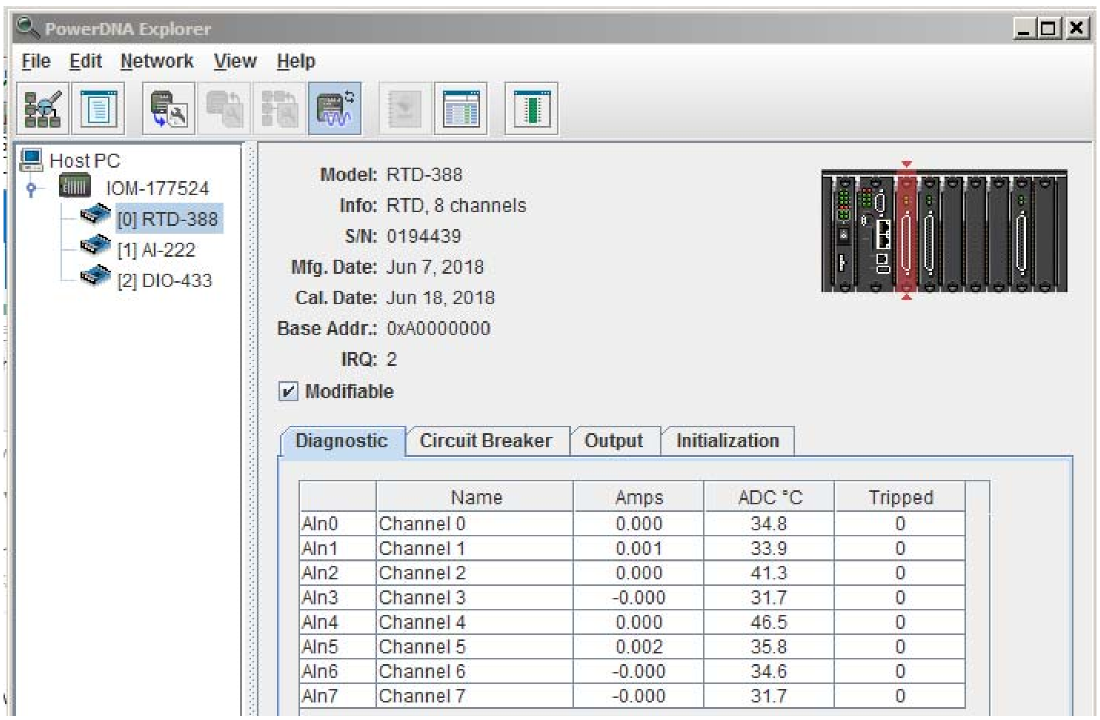

Figure 1-4 Diagnostic Display of the RTD-388 Board

1.8.1 Diagnostic Readback Values

The PowerDNA Explorer **Diagnostic** tab for the RTD-388 provides diagnostic readings and contains the following columns (see image above):

- **AInx**: The channel number
- **Name**: A name or note that you wish to give to the channel
- **Amps:** Guardian readback I_{in} current (in mA)
- **ADC °C**: Guardian on-die temperature of the ADC chip (°C)
- **Tripped**: Circuit breaker state (0 is not tripped; 1 is tripped)

To read diagnostic values, click the **Read Input Data** button in the menu bar. Note that reading diagnostic input values and circuit breaker status can take several seconds to update.

1.8.2 Circuit Breaker Values

The PowerDNA Explorer **Circuit Breaker** tab for the RTD-388 contains the following columns:

- **AOutx**: The channel number
- **Name**: A name or note that you wish to give to the channel
- **Tripped**: Circuit breaker tripped indicator
- **Reset**: Controls for immediately reengaging the circuit breaker if tripped

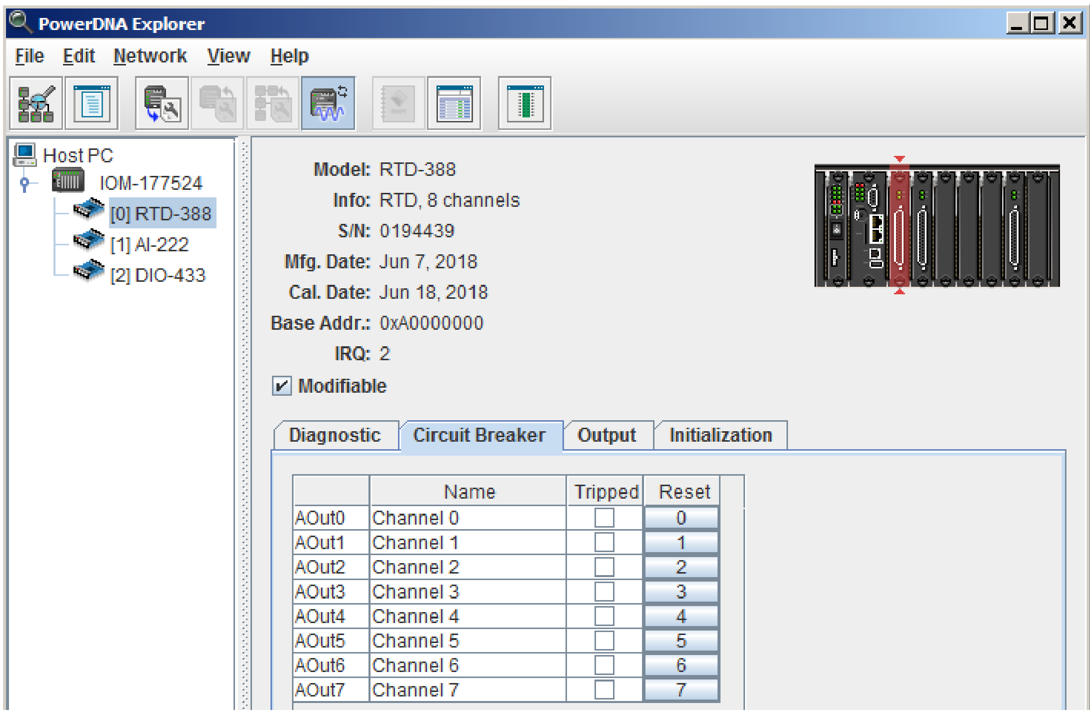

Figure 1-5 Circuit Breaker Display of the RTD-388 Board

1.8.3 Output Values

The PowerDNA Explorer **Output** tab contains the following columns:

- **AInx**: The channel number
- **Name**: A name or note that you wish to give to the channel
- **Ohms**: Displays currently set output resistance, and provides a textbox for entering new resistance (program RTD-388-1 from 180 Ω to 3900 Ω; RTD-388-100 from 23 Ω to 390 Ω)
- **Degrees**: Displays temperature that corresponds with the value set in **Ohms**. This temperature calculation is based on **Unit**, **αlpha**, and **R0(Ω)**. Also provides a textbox for entering a new output resistance as its corresponding RTD temperature.
- **Unit**: Pulldown menu of available temperature units
- **αlpha**: Pulldown menu of selections of RTD temperature coefficients of resistance (α constants). Select whichever α corresponds with the RTD you are simulating
- **R0(Ω)**: The RTD nominal resistance (for RTD-388, program 1000; for RTD-388-100, program 100)

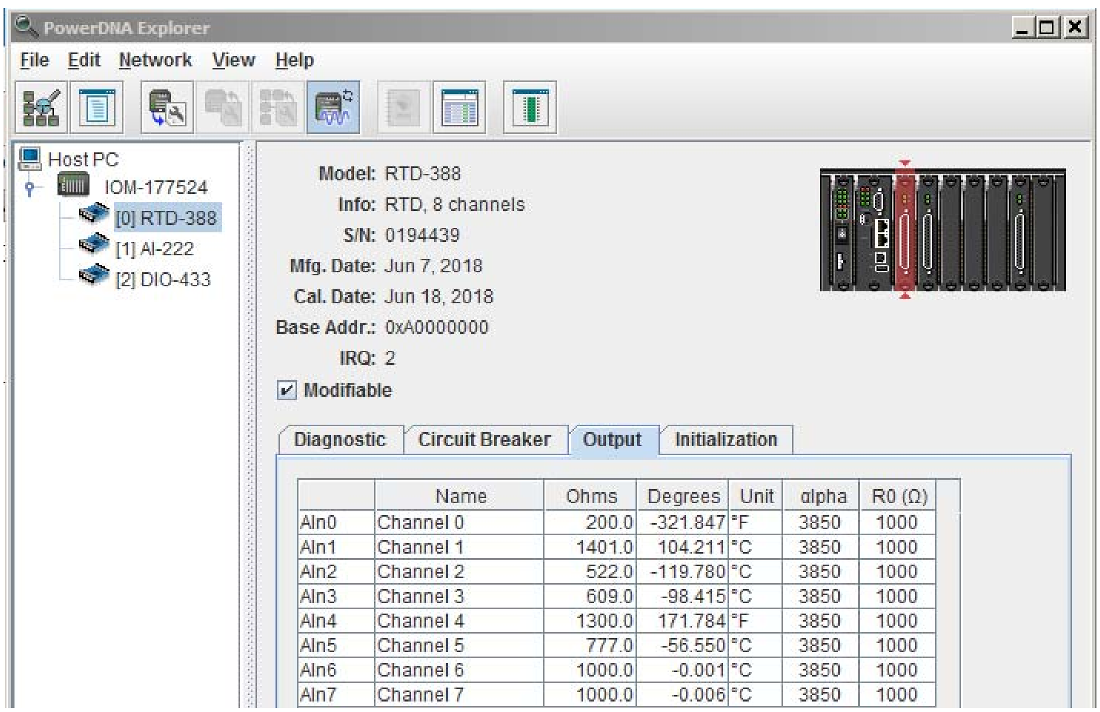

Figure 1-6 Output Display of the RTD-388 Board

The **Ohms** value represents the current resistance value on each RTD-388 channel. Values set in this display are written to hardware instantaneously after pressing **Enter** in the numeric field.

Users can choose to read and write RTD values in temperature instead of ohms by configuring the **Degrees, Unit, αlpha**, and **R0(Ω)** columns.

1.8.4 Initialization Values

The **Initialization** tab for the RTD-388 is similar to the immediate output but instead of outputting values immediately, stores values into the EEPROM configuration to set the output value on power-up. The factory default value is 1000 Ω for the RTD-388 and 100 Ω for the RTD-388-100.

Chapter 2 Programming with the High-Level API

This chapter provides the following information about using the UeiDaq high-level Framework API to program the DNx-RTD-388:

- About the High-level Framework (Section 2.1)
- Creating a Session (Section 2.2)
- Configuring the Resource String (Section 2.3)
- Configuring for Output (Section 2.4)
- Configuring the Timing (Section 2.5)
- Writing Output Data (Section 2.6)
- Monitoring Guardian Diagnostic Measurements (Section 2.7)
- Accessing Circuit Breakers (Section 2.8)
- Simulating Open or Short Circuits (Section 2.9)
- Cleaning-up the Session (Section 2.10)

2.1 About the High-level Framework

UeiDaq Framework is object oriented and its objects can be manipulated in the same manner from different development environments, such as Visual C++, Visual Basic, or LabVIEW.

UeiDaq Framework is bundled with examples for supported programming languages. Examples are located under the UEI programs group in:

- *Start » Programs » UEI » Framework » Examples*

The following sections focus on the C++ API, but the concept is the same no matter which programming language you use.

Please refer to the “*UeiDaq Framework User Manual*” for more information on use of other programming languages.

2.2 Creating a Session

The Session object controls all operations on your PowerDNx device. Therefore, the first task is to create a session object:

```
// create a session object for output
                CUeiSession rtdSimSession;
```

2.3 Configuring the Resource String

UeiDaq Framework uses resource strings to select which device, subsystem and channels to use within a session. The resource string syntax is similar to a web URL:

```
<device class>://<IP address>/<Device Id>/<Subsystem><Channel list>
```

For PowerDNA and RACKtangle, the device class is **pdna**.

For example, the following resource string selects analog output lines 0,1,2,3 on device 1 at IP address 192.168.100.2: “pdna://192.168.100.2/Dev1/Ao0:3” as a range, or as a list “pdna://192.168.100.2/Dev1/Ao0,1,2,3”.

2.4 Configuring for Output

You can configure the RTD-388 to program the output resistance in ohms or as RTD temperatures, which will be converted to ohms by the Framework.

2.4.1 Configure for Programming as RTD Temperature

The `CUeiSimulatedRTDChannel` class configures a simulated RTD channel to output the resistance corresponding to a given RTD temperature.

RTD sensors are resistive sensors whose resistance varies with temperature.

When creating a simulated RTD channel, you provide the type of RTD and temperature scale used for temperature-to-resistance conversion.

RTD types have specific alpha (α) constants, which are used to calculate the resistance from the temperature using the “Callendar Van-Dusen” equations. The α constant, also known as the temperature coefficient of resistance, defines the resistance change factor per degree of temperature change. The RTD type is used to select the proper coefficients A, B and C for the Callendar Van-Dusen equation.

Use the `CreateSimulatedRTDChannel()` method to configure one or more RTD channel(s). For example, the following call configures channel 0 through 7 of an RTD-388 set as device 5 to output simulated RTDs with 0.003850 α constants from temperatures programmed in Celsius:

```
// Create RTD simulation session with 8 channels

rtdSimSession.CreateSimulatedRTDChannel(
                              "pdna://192.168.100.2/dev5/ao0:7",
                              UeiRTDType3850, 1000.0,
                              UEITemperatureScaleCelsius);
```

`CreateSimulatedRTDChannel` configures the following parameters:

- **rtdType**: The type of RTD sensor to simulate:
  - `UeiRTDType3750`
  - `UeiRTDType3850`
  - `UeiRTDType3902`
  - `UeiRTDType3911`
  - `UeiRTDType3916`
  - `UeiRTDType3920`
  - `UeiRTDType3926`
  - `UeiRTDType3928`
  - `UeiRTDTypeCustom`
- **rtdNominalResistance**: The nominal resistance of the RTD at the ice point (0 degrees Celsius). (double)
  Enter 1000 for the RTD-388-1 and 100 for the RTD-388-100.
- **tempScale**: The temperature unit used to convert temperature to ohms:
  - `UEITemperatureScaleCelsius`
  - `UEITemperatureScaleFahrenheit`
  - `UEITemperatureScaleKelvin`
  - `UEITemperatureScaleRankine`

2.4.2 Configure for Programming in Ohms

The `CUeiAOChannel` class configures a simulated RTD channel to output the resistance as given in ohms.

Use the `CreateAOChannel()` method to configure one or more RTD channel(s).

For example, the following call configures channel 0 through 7 of an RTD-388 set as device 5 to output resistances in the range of 180 Ω (minimum) to 3900 Ω (maximum):

```
// Create RTD simulation session with 8 channels, but program in ohms

rtdSimSession.CreateAOChannel(
                           "pdna://192.168.100.2/dev5/ao0:7",
                           180.0, 3900.0);
```

2.5 Configuring the Timing

You can configure the RTD-388 to run in simple mode (point by point). The delay between samples is determined by software on the host computer.

The following sample shows how to configure the simple mode. Please refer to the "*UeiDaq Framework User's Manual*" to learn more about timing modes.

```
// configure timing of input for point-by-point (simple mode)

rtdSimSession.ConfigureTimingForSimpleIO();
```

2.6 Writing Output Data

Writing data is done using a *writer* object(s). The `CUeiAnalogScaledWriter` object programs output resistance in °C/F/K/R or in ohms.

Writing temperature data in degrees: your writer object will write data scaled as an RTD temperature if channels are configured as the `CUeiSimulatedRTDChannel` class (Section 2.4). Framework automatically performs a conversion using your RTD type and temperature scale.

Writing resistance data in ohms: your writer object will write data programmed as ohms if channels are configured as the `CUeiAOChannel` class (Section 2.4).

Framework generates binary code from the programmed value before sending the data to update the output resistance.

The following sample code shows how to create a scaled writer object and write a sample:

```
// create a writer and link it to the session's stream

CueiAnalogScaledWriter writer(rtdSimSession.GetDataStream());

// the buffer must be big enough to contain one value per channel

double data[2] = {0.0, 0.0};

// write one scan, where the buffer will contain one value per channel

writer.WriteSingleScan(data);
```

2.7 Monitoring Guardian Diagnostic Measurements

Applications can monitor Guardian diagnostic current and temperature measurements. Diagnostic measurements are read using the `CreateAIChannel` API, which returns current in amps and temperature in degrees Celsius.

Guardian diagnostic channels map as follows:

- Current measurements are read on diagnostic channels 16 to 23:
  AO channel 0 uses ai 16 to read current, AO channel 1 uses ai 17 to read current, etc.
- Internal ADC temperature measurements are read on diagnostic channels 32 to 39:
  AO channel 0 uses ai 32 to read temperature, AO channel 1 uses ai 33 to read temperature, etc.

NOTE: When entering the diagnostic channel numbers in the resource string, you must begin the channel list with “`ai`”, follow it by “`+`”, and then list the channel numbers.

The following code shows how to read diagnostic current and temperature on AO channels 0 through 1 for a RTD-388:

```
// create a Guardian input session to read current & temperature

CUeiSession guardianSession;
guardianSession.CreateAIChannel(
                "pdna://192.168.100.2/Dev1/Ai+16,32,17,33",
               -10.0, 10.0,
               UeiAIChannelInputModeDifferential);
```

```
// configure the Guardian input session for simple timing

guardianSession.ConfigureTimingForSimpleIO();

// the buffer must be big enough to contain one value per channel

CUeiAnalogScaledReader aiReader(guardianSession.GetDataStream());
double values[4];

// read one scan

aiReader.ReadSingleScan(values);
```

2.8 Accessing Circuit Breakers

The RTD-388 circuit breakers can be accessed with a `CUeiCircuitBreaker` object.

The first step in accessing the circuit breakers is to create a `CUeiCircuitBreaker` object:

```
// Create a circuit breaker and link it to the session's stream for
//   circuit breaker on analog output channel (channel 4 in this case)

CUeiCircuitBreaker cb(rtdSimSession.GetDataStream(), 4);
```

2.8.1 Reading Circuit Breaker Status

Call the `ReadStatus` method to retrieve the RTD-388 CB status masks.

The circuit breakers will trip when the default minimum or maximum conditions are exceeded:

- Current limits are approximately ±4.5 mA
- Temperature limit is 105° C

Each bit in the `current` status mask corresponds to a circuit breaker: 1 if CB is currently tripped, 0 otherwise.

Each bit in the `sticky` status mask corresponds to a circuit breaker: 1 if CB was tripped at least once since last time status was read, 0 otherwise.

```
// Read CB status

uInt32 currStatus, stickyStatus;
cb.ReadStatus(&currStatus, &stickyStatus);
```

2.8.2 Reseting the Circuit Breaker

Use the `Reset` method on a circuit breaker object to reset one or more breakers.

The mask parameter specifies which circuit breaker to reset (1 to reset, 0 to leave alone).

```
// Reset breakers for analog output channels 0 and 2

cb.Reset( 1<<0 | 1<<2 );
```

2.9 Simulating Open or Short Circuits

The RTD-388 simulated open or short circuit functionality can be accessed with a `CUeiCircuitBreaker` object.

The first step in simulating an open or short circuit on RTD-388 channels is to create a `CUeiCircuitBreaker` object:

```
// Create a circuit breaker object and link it to the session's stream
//   on any channel (channel 4 is arbitrarily chosen in this case)

CUeiCircuitBreaker cb(rtdSimSession.GetDataStream(), 4);
```

2.9.1 Enabling Open or Short Circuit Simulation

Call the `WriteOpenShortSimulation()` method to enable or disable an open or short circuit state on a RTD-388 CB channel. If both open and short circuit states are programmed, the short circuit has priority, and the channel will simulate a short.

`WriteOpenShortSimulation()` accepts a structure of type `tUeiAOShortOpenCircuitParameters`:

```
typedef struct _UeiAOShortOpenCircuitParameters
{
   unsigned int openBitmask;
   unsigned int shortBitmask;
   unsigned int shortDuration;
} tUeiAOShortOpenCircuitParameters;
```

where parameters are defined as follows:

- **`openBitmask`**: List of channels to open circuit.
  '1' in bit position 0 opens ch0, '1' in bit position 1 opens ch1, etc.
- **`shortBitmask`**: List of channels to short circuit.
  '1' in bit position 0 shorts ch0, '1' in bit position 1 shorts ch1, etc.:
- **`shortDuration`**: Maximum amount of time that an output will remain short circuited, in 100 µS increments.
  Defaults to 0, which results in the short circuit not opening automatically after a timed delay: control of disabling the short circuit is accomplished via the API call.

The following example simulates an open circuit on even channels and simulates a short circuit on odd channels:

```
// Simulating open and short circuits

tUeiAOShortOpenCircuitParameters aoOpShParam;
aoOpShParam.openBitmask = 0x55;
aoOpShParam.shortBitmask = 0xAA;
aoOpShParam.shortDuration = 0; // leave under app control

cbs.WriteOpenShortSimulation(aoOpShParam);
```

2.9.2 Disabling Open or Short Circuit Simulation

The `WriteOpenShortSimulation()` method is also used to disable an open or short circuit state on a RTD-388 CB channel.

The following example reverts channels to no longer simulate an open or short by setting bitmasks to 0:

```
// Set channels to output resistance instead of simulating open/short

aoOpShParam.openBitmask = 0x0;
aoOpShParam.shortBitmask = 0x0;

cbs.WriteOpenShortSimulation(aoOpShParam);
```

2.10 Cleaning-up the Session

The session object will clean itself up when it goes out of scope or when it is destroyed. To reuse the object with a different set of channels or parameters, you can manually clean up the session with the `CleanUp` call as follows:

```
// clean up the sessions

guardianSession.CleanUp();
rtdSimSession.CleanUp();
```

Chapter 3 Programming with the Low-Level API

This chapter provides the following information about programming the RTD-388 using the low-level API:

- About the Low-level API (Section 3.1)
- Low-level Functions (Section 3.2)
- Low-level Programming Techniques (Section 3.3)
  - Data Transfer Modes (Section 3.3.1)
- Programming the RTD-388 (Immediate Mode) (Section 3.4)
  - Configuring Resistance Channels (Section 3.4.1)
  - Writing Output Resistance (Section 3.4.2)
  - Simulating Open Circuits (Section 3.4.3)
  - Simulating Short Circuits (Section 3.4.4)
  - Configuring Guardian Diagnostics (Section 3.4.5)
  - Reading Guardian Diagnostic Values & Timestamp (Section 3.4.6)
  - Reading Circuit Breaker Status & Reengaging (Section 3.4.7)

3.1 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, however, we recommend that you use the UeiDaq high-level Framework API (see **Chapter 2**). The Framework extends the low-level API with additional functionality that makes programming easier and faster, and 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 systems:
  <PowerDNA-x.y.z>/docs
- On Windows systems:
  Start » All Programs » UEI » PowerDNA » Documentation

3.2 Low-level Functions

Table 3-1 provides a summary of RTD-388-specific functions. All low-level functions are described in detail in the *PowerDNA API Reference Manual*.

Table 3-1 Summary of Low-level API Functions for DNx-RTD-388

Function	Description
`DqAdv388Write`	Configures RTD-388 resistance values. Users provide an array of resistances and the API converts ohms to equivalent encoded resistance data. This API can optionally immediately write the encoded resistance data directly to UEI hardware or pass back an array of encoded resistance data that can later be written for programming hardware using `DqAdv388WriteBin`.
`DqAdv388WriteBin`	Writes encoded resistance data to UEI hardware. Alternative method of updating resistance values: This API can be used after `DqAdv388Write`, and accepts the array of encoded resistance data that is optionally returned from `DqAdv388Write`.
`DqAdv388WriteDiag`	Controls RTD-388 simulated open- and short-circuit diagnostic features.
`DqAdv388ADCEnable`	Enables Guardian ADC for diagnostic readings.
`DqAdv388ReadADC`	Read back current and/or temperature from Guardian diagnostic ADCs.
`DqAdv388Reengage`	Reset tripped circuit breakers for selected channels.
`DqAdv388CBStatus`	Get the circuit breaker status for selected channels.
`DqAdv388SetConfig`	Advanced configuration of circuit breakers and ADCs.

3.3 Low-level Programming Techniques

Application developers are encouraged to explore existing source code examples when first programming the RTD-388. Sample code provided with the installation is self-documented and serves as a good starting point.

Code examples are located in the following directories:

- On Linux systems: <PowerDNA-x.y.z>/src/DAQLib_Samples
- On Windows: *Start » All Programs » UEI » PowerDNA » Examples*

Sample code has the name of the I/O boards being programmed embedded in the sample name.

For example, `Sample388` contains sample code for running the an RTD-388 in Immediate mode and reading Guardian diagnostics. It also contains a `rtdconversion.h` conversion file, that provides API for converting resistances to temperature, (e.g., `UeiDaqConvertArrayOfRtdResistancesToTemperature()`).

3.3.1 Data Transfer Modes

The RTD-388 supports the following acquisition modes.

Immediate (point-to-point): Transfers a single data point per channel of a single I/O board at a non-deterministic pace.
Runs at a maximum of 100 Hz.

RtDMap: Transfers samples as specified in a user-defined map of I/O boards and channels. The timebase is maintained by the host application. RtDMap delivers 1 data sample per channel

NOTE: API that implement data acquisition modes and additional mode descriptions are provided in the *PowerDNA API Reference Manual*.

3.4 Programming the RTD-388 (Immediate Mode)

The following sections provide an overview of how to set up and use your RTD-388 in Immediate Mode using the low-level API.

For best results, use this overview in conjunction with actual sample code, (e.g, Sample388). This overview does not address all initialization or error handling. Refer to Section 3.3 above for sample code location.

3.4.1 Configuring Resistance Channels

Users declare a channel list array and initialize the list of RTD-388 channels to enable and output samples to.

```
uint32 CL[CHANNELS]; //  CHANNELS is max of 8
```

You can set up the channel list to enable channels sequentially or in whichever order you choose:

```
// to order channels sequentially in the channel list:
for (i = 0; i < CHANNELS; i++) {
      CL[i] = i;
}
```

3.4.2 Writing Output Resistance

In Immediate mode, use the `DqAdv388Write()` and optionally the `DqAdv388WriteBin()` APIs to write to the RTD-388 and update the resistance for each of the enabled channels (in the order of the channel list).

NOTE: The `DqAdv388Write()` can either write the data words to hardware immediately or pass back an array of resistance-encoded data. If a `NULL` is passed as the return array, then the `DqAdv388Write()` API will program the hardware directly.

Write encoded resistance values immediately to hardware:

```
// RL array is programmed in the order of the CL array
uint32 RL[CHANNELS]; //

//    For example, resistance RL[0] will set channel CL[0]
for (i = 0; i < CHANNELS; i++) {
         RL[i] = resistanceToProgram(); //RTD-388: values from 180Ω to 3900Ω
                                        //RTD-388-100: 23Ω to 390Ω
}
// pass arrays of channels and resistance values.
//   API converts resistance value to encoded resistance data for
//   updating hardware on each channel in CL to the corresponding RL
//   resistance value.
// When a NULL is passed as the last parameter, output hardware is updated
//   immediately
DqAdv388Write(hd,            // handle to IOM
              DEVN,          // position RTD-388 inserted in the chassis
              CHANNELS,      // total number of channels enabled
              CL,            // channel list configured previously
              RL,            // list of resistances to program CL
              0,             // 8-bit delay in 12.5us increment
              0,             // flags, <Reserved> set to 0
              NULL);         // immediately update output resistance
```

3.4.2.1 Alternate Convert and Later Write Output Resistances

Optionally, as an alternative, users can use the `DqAdv388WriteBin()` API to write the hardware later.

If an array (`binvals` in the code example below) is passed as the last parameter of the `DqAdv388Write` API, the API will not write the hardware, but instead pass back a list of encoded resistance values that can be used later with the `DqAdv388WriteBin` API to write hardware.

```
// use DqAdv388Write to return “binvals”, which is a list
//   of resistance encoded data for updating hardware at a later time on
//   each channel in CL to the corresponding RL resistance value
uint32 CL[CHANNELS]; //  CHANNELS is max of 8

DqAdv388Write(hd,           // handle to IOM
              DEVN,         // position RTD-388 inserted in the chassis
              CHANNELS,     // total number of channels enabled
              CL,           // channel list configured previously
              RL,           // list of resistances to program CL
              0,            // 8-bit delay in 12.5us increment
              0,            // flags, <Reserved> set to 0
              binvals);     // return array of encoded resistance data
```

To write encoded resistance data to hardware (update output resistance), do the following:

```
DqAdv388WriteBin(hd,
              DEVN,
              CHANNELS,
              binvals);
```

3.4.3 Simulating Open Circuits

Use the `DqAdv388WriteDiag()` API to open the Simulated Open Circuit relay (refer to Figure 1-1 on page 7 for location).

The `ch_open` parameter is a bitmask of channels to open-circuit. A ‘1’ in the bit position of the channel number causes an open circuit on that channel (a ‘0’ closes the relay).

The following code snippet will cause an open circuit on channels 1, 3, and 4.

```
uint32 ch_open = 0x1A;

DqAdv388WriteDiag(hd,       // handle to IOM
              DEVN,         // position RTD-388 inserted in the chassis
              ch_open,      // bitmask of channels to open
              0,            // no short-circuit simulation
              0);           // not used here; for short-circuit simulation
```

The following code snippet will close all of the Simulated Open Circuit relays:

```
uint32 ch_open = 0x0;
DqAdv388WriteDiag(hd,       // handle to IOM
              DEVN,         // position RTD-388 inserted in the chassis
              ch_open,      // bitmask of channels to open
              0,            // no short-circuit simulation
              0);           // not used here; for short-circuit simulation
```

3.4.4 Simulating Short Circuits

Use the DqAdv388WriteDiag() API to close the Simulated Short Circuit relay (refer to Figure 1-1 on page 7 for location).

The ch_short parameter is a bitmask of channels to short-circuit. A ‘1’ in the bit position of the channel number causes an short circuit on that channel (a ‘0’ opens the relay).

You also program a maximum duration the short circuit will last (programmed in 100 µs intervals).

The following code snippet will cause an short circuit on channels 0, 3, and 5.

```
uint 32 ch_short = 0x29;
DqAdv388WriteDiag(hd,       // handle to IOM
                DEVN,       // position RTD-388 inserted in the chassis
                0,          // no open-circuit simulation
                ch_short,   // bitmask of channels to short
                0);         // The short circuit will happen immediately
```

The following code snippet will open all of the Simulated Short Circuit relays:

```
uint32 ch_short = 0x0;
DqAdv388WriteDiag(hd,       // handle to IOM
                DEVN,       // position RTD-388 inserted in the chassis
                0,          // no open-circuit simulation
                ch_short,   // bitmask of channels to remove short circuit
                0);         // not used here;
```

3.4.5 Configuring Guardian Diagnostics

UEI provides access to reading diagnostic Guardian current and ADC temperature, as well as timestamp data.

To read diagnostic data, you must enable the ADC with the DqAdv388ADCEnable() API, and then configure channels and read data with the DqAdv388ReadADC() API.

Guardian diagnostic readings are defined as follows:

Diagnostic Channel	Description
DQ_RTD388_I_IN_CH_0 to DQ_RTD388_I_IN_CH_7	I_{in} current for channels 0 through 7
DQ_RTD388_T_CH_0 to DQ_RTD388_T_CH_7	On-die ADC temperature for channels 0 through 7

Table 3-2. RTD-388 Diagnostic Measurements

Define the number of Guardian/Timestamp channels to read:

```
//  There are 2 Guardian channels (Iin and Temperature)
//   for each of the enabled output channels (CHANNELS)

#define TIMESTAMP    (0)          // no timestamp included for this example
#define RCHANNELS    ((2 * CHANNELS) + TIMESTAMP)
```

Declare a read array:

```
uint32 rcl[RCHANNELS];      // create a channel list of readback data
```

Create Guardian/Timestamp channel list:

Create Guardian diagnostic channel list using the `#define` values in Table 3-2.

To set up a channel list for reading current and temperature diagnostic data for channel 0 and channel 1, you can do the following:

```
rcl[0] = DQ_RTD388_I_IN_CH_0; // read diagnostic Iin on channel 0
rcl[1] = DQ_RTD388_T_CH_0;    // read diagnostic Temperature on channel 0
rcl[2] = DQ_RTD388_I_IN_CH_1; // read diagnostic Iin on channel 1
rcl[3] = DQ_RTD388_T_CH_1;    // read diagnostic Temperature on channel 1
```

NOTE: The `#define` values for each channel increment sequentially. For example, for channel 1, #define equivalents are

- `DQ_RTD388_I_IN_CH_1` equals `(DQ_RTD388_I_IN_CH_0)`+1
- `DQ_RTD388_T_IN_CH_1` equals `(DQ_RTD388_T_IN_CH_0)`+1

Enable ADC:

Use the `DqAdv388ADCEnable` API to enable the ADCs on configured channels. Note that the ADC is enabled on power up.

```
//   mask=0x1 to configure channel 0; mask =0xff to configure all 8 channels

mask=0x3;                   //configure channel 0 & 1

DqAdv388ADCEnable(hd,       // handle to IOM
                DEVN,       // position RTD-388 inserted in the chassis
                TRUE,       // TRUE to enable, FALSE to disable
                &mask);     // pointer to uint32* mask (channels to enable)
                            // API returns which chs are enabled/disabled
```

Configure ADC hardware:

Configure hardware with the first call to `DqAdv388ReadADC()` API. The first read is for hardware configuration only.

```
//   Pass zeros instead of return arrays for the last 2 parameters
//   since no data is returned in this first call for config only

DqAdv388ReadADC(hd,         // handle to IOM
                DEVN,       // position RTD-388 inserted in the chassis
                RCHANNELS, // total number of input (diagnostic channels)
                rcl,        // read channel list configured in previous step
                0,          // set to 0 for config
                0);         // set to 0 for config
```

3.4.6 Reading Guardian Diagnostic Values & Timestamp

The first call to the `DqAdv388ReadADC()` API is to program the ADC hardware, and then subsequent calls to `DqAdv388ReadADC()` read diagnostic data values (in the order of the `rcl` read channel list):

Declare arrays to hold returned measurements:

```
uint32 b_adcdata[CHANNELS*2 + TIMESTAMP];
double f_adcdata[CHANNELS*2 + TIMESTAMP];
```

Read Guardian diagnostic (& optional timestamp):

```
DqAdv388ReadADC(hd,         // handle to IOM
                DEVN,      // position RTD-388 inserted in the chassis
                RCHANNELS, // total number of diagnostic channels enabled
                rcl,       // read channel list
                b_adcdata, // array for returned raw readback data
                f_adcdata);// array for returned floating pt readback
```

3.4.7 Reading Circuit Breaker Status & Reengaging

By default, the circuit breakers on each output channel are enabled. Circuit breakers are programmed to trip when the current or temperature limit is exceeded.

The default limits are set to the following:

- Current limits are approximately ±4.5 mA
- ADC internal temperature limit is 105°C

To monitor the status of the circuit breakers, use the `DqAdv388CBStatus()` API, and to reengage circuit breakers, use the `DqAdv388Reengage` API.

Declare an array to hold circuit breaker status values:

```
//   cbstatus[0] is bitmask of all channels; cbstatus[1:8] are individual ch
uint32 cbstatus[9]; // all 8 channels plus 1
```

Read circuit breaker status:

```
// The channel mask set as 0x1 reads 1st channel status;
//     0xff reads all 8 channels' status
DqAdv388CBStatus(hd, DEVN, mask, 0, 0, cbstatus);
```

- `cbstatus[0]:`
  bit 31:24 - current state of simulated short circuit for ch 7:0 (1 is shorted)
  bit 23:16 - current status of CB for ch 7:0 (1 is tripped)
  (note that bits 15:0 are sticky; cleared on read)
  bit 15:8 - indicates ADC min/max value has been evaluated for ch 7:0
  bit 7:0 - sticky indicator that ch 7:0 CB had been tripped since last read
- `cbstatus[1:8]:`
  refer to *PowerDNA API Reference Manual*

Reengage circuit breakers:

```
// pass the DqAdv388Reengage API a bitmask of channels to reengage
uint32 resetAllChannels = 0xff;

if ((yourCriteria == TRUE) && ((cbstatus[0]>>16) & 0xff)){
    DqAdv388Reengage(hd, DEVN, resetAllChannels);
}
```

NOTE: Users can alternatively read the CB status using the `DqAdv388ReadADC` API described in Section 3.4.6.
Bit 30 of the raw readback data (`b_adcdata`) that is passed back with the `DqAdv388ReadADC` API provides the state of the circuit breaker: A '1' in bit 30 means the CB for that channel is tripped; a '0' means it is not tripped.

Appendix A

A.1 Accessories

The following cables and STP boards are available for the RTD-388 board.

DNA-CBL-37

This is a 37-conductor flat ribbon cable with 37-pin male D-sub connectors on both ends. The length is 3ft and the weight is 3.4 ounces or 98 grams.

DNA-CBL-37S

This is a 37-conductor round shielded cable with 37-pin male D-sub connectors on both ends. It is made with round, heavy-shielded cable; 3 ft (90 cm) long, weight of 10 ounces or 282 grams; also available in 10ft and 20ft lengths.

DNA-STP-37

The DNA-STP-37 provides easy screw terminal connections for all DNx series I/O boards which utilize the 37-pin connector scheme.

The DNA-STP-37 is connected to the I/O board via either DNA-CBL-37 or DNA-CBL-37S cable.

The dimensions of the STP-37 board are 4.2w x 2.8d x 1.0h inch or 10.6 x 7.1 x 7.6 cm (with standoffs). The weight of the STP-37 board is 2.4 ounces or 69 grams.

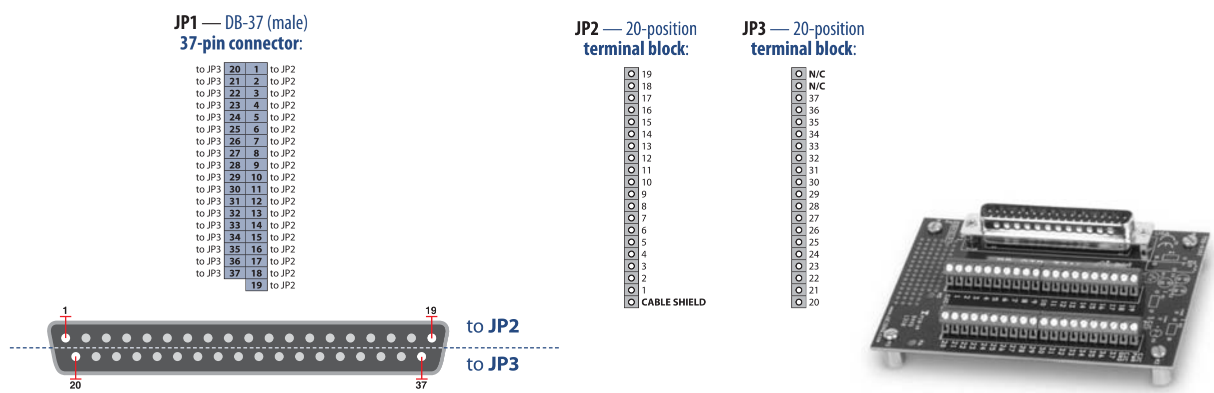

Figure A-1 Pinout and photo of DNA-STP-37 screw terminal panel

Index