United Electronic Industries

The High-Performance Alternative

DNx-TC-378
—
User Manual

TC-378 8-channel, 16-bit, isolated thermocouple simulator board
with built-in-test
for the PowerDNA Cube and RACK series chassis

May 2018

PN Man-DNx-TC-378

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 manual outlines the feature set and use of the DNx-TC-378, a fully isolated, 8-channel, high-precision thermocouple simulator board.

The following sections are provided in this chapter:

- Organization of this Manual (Section 1.1)
- TC-378 Board Overview (Section 1.2)
- Features (Section 1.3)
- Specification (Section 1.4)
- Device Architecture (Section 1.5)
- Indicators (Section 1.6)
- Wiring & Connections (pinout) (Section 1.7)
- PowerDNA Explorer for TC-378 (Section 1.8)

1.1 Organization of this Manual

This *DNx-TC-378 User Manual* is organized as follows:

- **Introduction**
  Chapter 1 provides an overview of DNx-TC-378 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 TC-378 series board.
- **Appendix A - Accessories**
  The appendix provides a list of accessories available for use with the DNx-TC-378 board.
  Note that UEI offers a screw terminal panel (DNA-STP-378) with 3 built-in cold-junction compensation temperature sensors, specifically designed for use with the TC-378.
- **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.

Examples of Manual Conventions

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.

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 TC-378 Board Overview

The DNx-TC-378 boards are fully isolated, high-precision, 8-channel thermocouple simulator boards.

DNA-TC-378, DNR-TC-378, and DNF-TC-378 board versions are compatible with the UEI Cube, RACKtangle, and FLATRACK chassis respectively. These board versions are electronically identical and differ only in 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 DAC Output Capabilities

The TC-378 boards offer full 16-bit resolution and guarantee monotonicity over the entire operating temperature range. Each TC-378 channel provides an output range of ±125 mV. Each output is capable of driving up to ±10 mA (max). A shorted thermocouple can be simulated by setting the output voltage to 0 V.

1.2.2 Guardian Diagnostic Support

The DNx-TC-378 provides extensive built-in-test diagnostics.

An on-board A/D converter on each channel allows the user to monitor current, temperature, and output voltage. A solid state relay on each output allows the D/A channel to be disconnected from the field I/O, causing an open thermocouple to be simulated and allowing the input systems open TC detection circuitry to be tested.

1.2.3 Data Update Rates

A 1024 sample FIFO allows each D/A to be updated at 1 kHz per channel (8 kHz aggregate) without data loss.

1.2.4 Cold-junction Compensation (CJC)

The board provides three cold-junction input channels, allowing the TC-378 to measure temperature where the TC-378 is connected to the thermocouple input device of the application. This cold-junction temperature measurement can be used by the application software to compensate for errors caused by the lack of an actual cold-junction error (as there will not be the extraneous dissimilar metal connections, which in actual thermocouples cause the need for compensation).

1.2.5 Isolation

The outputs of the TC-378 are fully isolated from each other, and CJC channels are isolated as a group from the rest of the board. TheTC-378 board is isolated from the I/O chassis, as well as from other I/O boards within the I/O chassis. All connections are made through a 37-pin D connector.

1.2.6 Software Support

Software included with the DNx-TC-378 provides a comprehensive yet easy to use API that supports all popular operating systems including Windows, Linux, QNX, VxWorks and other 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 languages, such as C, C++, C#, VB.NET and scientific software packages such as LabVIEW and MATLAB.

1.3 Features

The TC-378 boards have the following features:

- 8 independent fully isolated outputs
- I^2C interface for reading up to 3 ADT7420 cold-junction temperature sensor inputs, allowing cold-junction compensation (if required)
- 1 kHz per channel max update rate
- ±125 mV output range supports all common thermocouples
- Guardian series diagnostics provide output voltage, temperature and current readback
- Operates in voltage or simulated thermocouple modes
- Simulates open thermocouples
- Tested to withstand 5g vibration, 50g shock, -40 to +85°C temperature, and altitude up to 70000 ft or 21000 meters.

1.4 Specification

The technical specification for the DNx-TC-378 boards are listed below.

Table 1-1. DNx-TC-378 Technical Specifications (at 25°C)

Analog Outputs	8 channels
Resolution	16-bits
Output range	±100 mV
Max Update Rate:	1 kHz/channel (8 kHz max aggregate)
FIFO Buffer Size	1024 samples
Output Accuracy	see table below for accuracy in °C
INL (no load)	±3 LSB (0.018%) (typical)
DNL (no load)	±1 LSB (0.006%) (typical)
Monotonicity	16 bits guaranteed over temperature
Gain Calibration Error	±10 µV, typ, ±20 µV, max
Offset Calibration Error	±5 µV, typ, ±10 µV, max
Output Impedance	<0.5 Ω (typ)
Current Drive	±10 mA/channel
Settling Time	1 ms to 16 bits
Power up state	0 V ±1 mV
Open TC resistance	1 M Ω minimum
Diagnostic readback	
Voltage range	±2 V
Voltage accuracy	±500 µV
Current range	±20 mA
Current accuracy	±100 µA
CJC Monitoring	
CJC Sensor type	ADT 7420, (included on DNA-STP-378)
CJC sensor accuracy	±0.1 °C typical, ±0.35 °C max (-40 °C to +105 °C)
Sample/Update rate	10 Hz
Isolation	350Vrms channel-to-channel and field wiring to chassis.
Power Consumption	4.0 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, 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
MTBF	>200,000 hours

Thermocouple Type	**Typical Error[1] at 0 °C** **(CJC 25°C)**	**Error[1] at 0 °C** **(CJC -20 to 85°C)**
E	±0.4	±0.9
J	±0.5	±1.0
K	±0.6	±1.2
N	±1.0	±1.8
R	±3.6	±6.0
S	±3.6	±6.0
T	±0.6	±1.2

Table 1-2. DNx-TC-378 accuracy in °C when used with DNA-STP-378

[1]Error Includes: Output measurement error, Error due to linearization math (when using UEI software) and CJC measurement error.

1.5 Device Architecture

Figure 1-1 is a block diagram of the architecture of the TC-378 board.

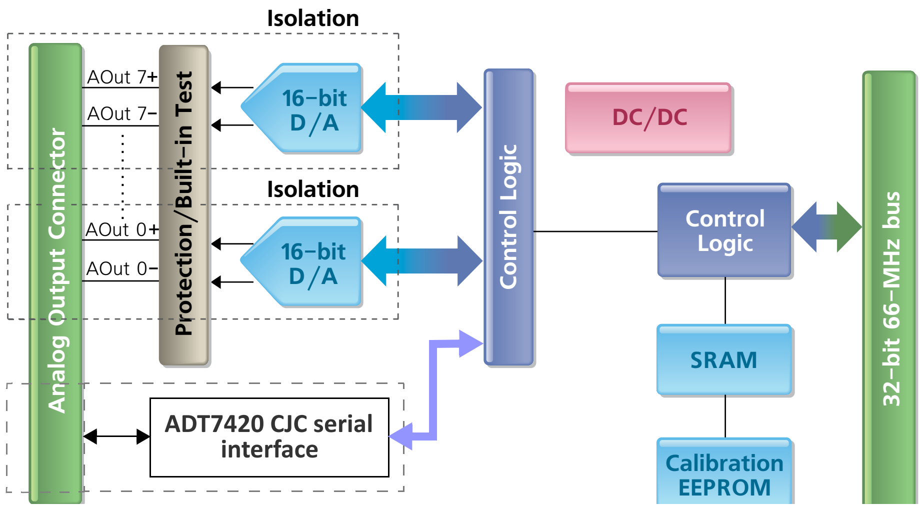

Figure 1-1 Block Diagram of the TC-378

1.5.1 TC-378 Output Circuitry

The TC-378 has eight individual analog output channels.

Each output channel includes a dedicated 16-bit D/A converter and a dedicated A/D converter to monitor voltage, current, and/or temperature for Guardian diagnostic functions (**Figure 1-2**). Channels are also equipped with a dedicated circuit breaker and an output relay to connect or disconnect the output to or from the load.

Each output channel is isolated from other channels, and the TC-378 board itself is also optically isolated from the other boards in the chassis.

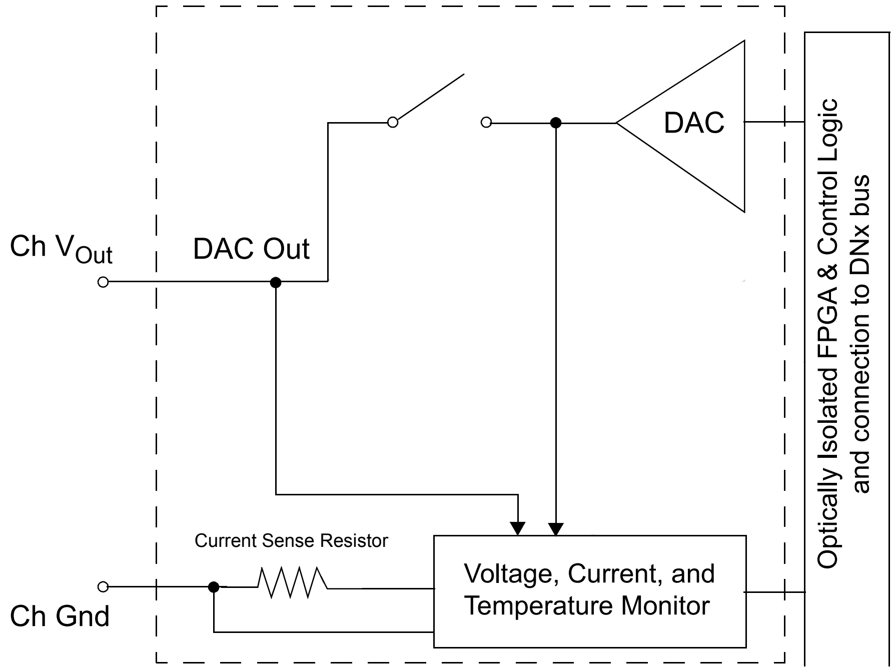
DAC
Ch V_Out
DAC Out
Current Sense Resistor
Voltage, Current, and Temperature Monitor
Ch Gnd
Optically Isolated FPGA & Control Logic and connection to DNx bus

Figure 1-2 Simplified Block Diagram of an TC-378 Channel

1.5.2 Guardian Diagnostic Measurements

Guardian diagnostic readings are performed by an analog-to-digital converter in the “Voltage, Current, and Temperature Monitor” block, (block diagram shown in **Figure 1-2**).

Each of the eight TC-378 output channels is equipped with its own Guardian ADC. The following measurements can be read as analog input values on the diagnostic ADC channels:

ADC Channel	TC-378 Diagnostic Measurement
0	Current through a shunt resistor between the output connector and A_{GND}
1	DAC output voltage after the relay
2	ADC on-die temperature sensor

Table 1-3. TC-378 Diagnostic Channels

1.5.2.1 Configuring Guardian Diagnostic Readings

The TC-378 Guardian ADC hardware consists of a 16-bit precision, sequential ΔΣ analog-to-digital converter, which is controlled by the on-board logic.

The sample rate of the Guardian ADC can be set to 5, 7.5 (default), 10, 20, 30, 40, 50 Hz.

UEI recommends using a low sampling rate (the default) when measuring diagnostic channels to allow the sequential ADC sufficient time to propagate through all measurements. Higher sampling frequencies yield higher errors.

1.5.3 Circuit Breakers

A relay at the output of the digital-to-analog converter acts as a circuit breaker, (refer to **Figure 1-2** for location).

By default, the circuit breakers are enabled. Control logic compares the current and temperature measurements to the minimum and maximum limits and trips the breaker if those limits are exceeded.

The following are the default minimum and maximum values:

- Current limits are ±8 mA
- ADC internal temperature limits are -45°C to 105°C

Developers programming using the low-level API (see **Chapter 3**) can customize circuit breaker functionality: the relay can be controlled manually or by the on-board logic, and minimum and maximum limits can be individually programmed.

Also, when programming with the low-level API, users can configure which Guardian measurements are used as trip criteria (up to 2 of the 3 Guardian measurements can be selected).

If a circuit breaker trips, you can disable the circuit breaker channel and/or issue a reset command to the circuit breaker once the signal returns within the min/max bounds.

1.5.4 Cold-junction Measurements

The TC-378 provides hardware support for cold-junction compensation (CJC).

The TC-378 incorporates an I^2C interface for acquiring temperature readings of up to three ADT7420 temperature sensors. This allows users to measure the temperature where the TC-378 is connected to the thermocouple input device of the application.

The user application can use acquired cold-junction temperature measurements to compensate for the lack of an actual cold-junction error inherent to thermocouple devices (normally caused by the dissimilar metal connections of an actual thermocouple device).

UEI offers a screw terminal panel accessory (DNA-STP-378) with three built-in ADT7420 sensors. The ADT7420 sensors have 16-bit ±0.25°C accuracy and can provide data over the I^2C bus on any of the three CJC I^2C lines to the main controller of the TC-378 board.

1.6 Indicators

The DNx-TC-378 indicators are described in **Table 1-4** and illustrated in **Figure 1-3**.

Table 1-4 TC-378 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

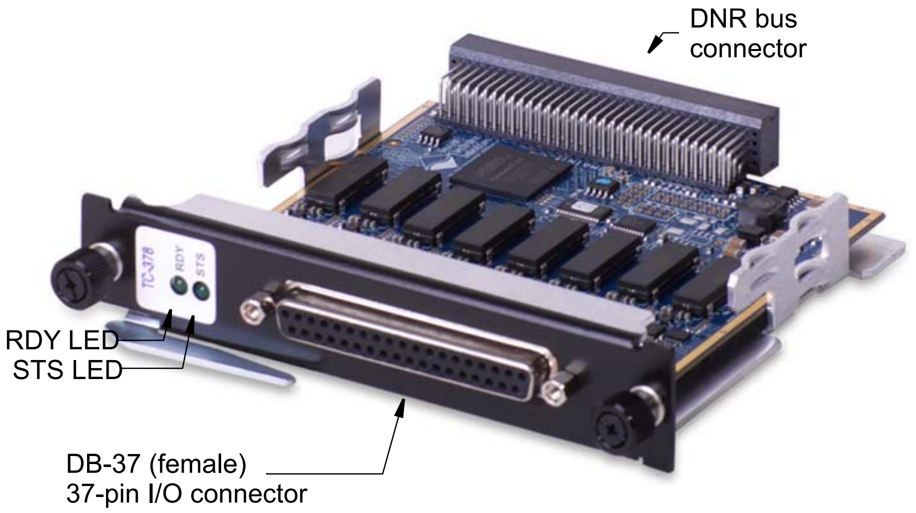

Figure 1-3 Photo of DNR-TC-378 Thermocouple Simulator Board

1.7 Wiring & Connections (pinout)

Figure 1-4 below illustrates the pinout of the TC-378.

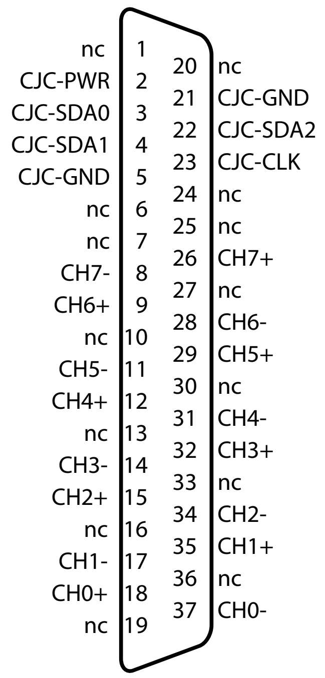

Figure 1-4 Pinout diagram of TC-378

NOTE: Do not connect to any pins marked as nc.

1.8 PowerDNA Explorer for TC-378

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 a TC-378 board, note that the right-hand panel contains five tabs:

- **Input**: displays diagnostic readback values
- **Output**: sets output levels immediately
- **Initialization**: initializes output levels applied at power-up
- **Shutdown**: sets output level applied at shutdown

1.8.1 Diagnostic Readback (Input) Values

The PowerDNA Explorer **Input** tab for the TC-378 provides diagnostic readings and contains the following columns:

- **AIn*x***: the channel number.
- **Name**: a name or note that you wish to give to the channel.
- **Subsequent columns** display the 3 Guardian diagnostic readback values as described in Section 1.5.2 and the circuit breaker state.

In PowerDNA Explorer diagnostic channels display as follows:

ADC Ch	Selector	Input Units
0	Current [mA]	mA
1	Output Voltage	Vo
2	Temp. [°C]	°C

The final column appends a “tripped” state if the circuit breaker was tripped.

To read diagnostic values, click the **Read Input Data** button. Note that reading diagnostic input values and circuit breaker status can take from several seconds to several minutes and may affect output readings.

1.8.2 Output Values

The PowerDNA Explorer **Output** tab for the TC-378 contains the following columns:

- **AOutx**: the channel number.
- **Name**: a name or note that you wish to give to the channel.
- **Value**: controls for immediately setting the voltage of the corresponding output channel.
  The valid output range (+/- 0.125 V) is shown in the **Output Range** display under the **Modifiable** checkbox.
  Values set in this display are written instantaneously when the control’s focus is released or after pressing **Enter** in the numeric field; pressing the **Store Config** button is not required for updating the value in hardware.

1.8.3 Initialization Values

The **Initialization** tab for the TC-378 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 0.

1.8.4 Shutdown Values

The **Shutdown** tab for the TC-378 is similar to the immediate output but instead of outputting values immediately, stores values into the EEPROM configuration to set the output value in shutdown mode. The factory default value is 0.

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-TC-378:

- 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)
- Accessing Circuit Breakers (Section 2.5)
- Configuring the Timing (Section 2.6)
- Writing Output Data (Section 2.7)
- Monitoring Guardian Diagnostic & CJC Measurements (Section 2.8)
- Cleaning-up the Session (Section 2.9)

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 tcSimSession;
```

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

The `UeiSimulatedTCChannel` class configures a simulated thermocouple channel to output the voltage corresponding to a given thermocouple temperature.

The type of thermocouple and temperature scale used for temperature-to-voltage conversion are entered as parameters. Users can also identify whether to offset the values with a measured CJC value.

Use the `CreateSimulatedTCChannel()` method to configure one or more channel(s). For example, the following call configures channel 0 through 7 of an TC-378 set as device 5 to output simulated thermocouple type K voltages:

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

tcSimSession.CreateSimulatedTCChannel(
                            "pdna://192.168.100.2/dev5/ao0:7",
                            UeiThermocoupleTypeK,
                            UEITemperatureScaleCelsius,
                            true); // enableCjc
```

`CreateSimulatedTCChannel` configures the following parameters:

- **Thermocouple Type**: The type of thermocouple to simulate:
  - `UeiThermocoupleTypeB`
  - `UeiThermocoupleTypeJ`
  - `UeiThermocoupleTypeK`
  - `UeiThermocoupleTypeN`
  - `UeiThermocoupleTypeT`
  - `UeiThermocoupleTypeE`
  - `UeiThermocoupleTypeR`
  - `UeiThermocoupleTypeS`
  - `UeiThermocoupleTypeC`
- **Temperature Scale**: The temperature unit used to convert temperature to volts:
  - `UEITemperatureScaleCelsius`
  - `UEITemperatureScaleFahrenheit`
  - `UEITemperatureScaleKelvin`
  - `UEITemperatureScaleRankine`
- **Enable CJC**: enable cold junction compensation: `true, false`

2.5 Accessing Circuit Breakers

The TC-378 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(tcSimSession.GetDataStream(), 4);
```

2.5.1 Reading Circuit Breaker Status

Call the `ReadStatus` method to retrieve the TC-378 CB status masks.

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

- Current limits are ±8 mA
- ADC internal temperature limits are -45° C to 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);
```

2.5.2 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 );
```

2.6 Configuring the Timing

You can configure the TC-378 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)

tcSimSession.ConfigureTimingForSimpleIO();
```

2.7 Writing Output Data

Writing data is done using a *writer* object(s).

Using the `CueiAnalogScaledWriter,` you can create a writer object that writes data scaled as temperature. Framework automatically performs a conversion using your thermocouple type and temperature scale. If CJC is enabled, Framework automatically reads the CJC channels and adjusts the voltage of the simulated TCs accordingly. Framework generates binary code from this value before sending the data to the D/A converter.

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(tcSimSession.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);
```

2.8 Monitoring Guardian Diagnostic & CJC Measure-ments

Applications can monitor Guardian diagnostic output voltage, current, and temperature measurements, as well as up to 3 cold-junction measurements:

- Diagnostic and CJC measurements can be read using the `CreateAIChannel` API, which returns current in amps, voltage in volts, temperature in degrees Celsius, and CJC values.
(refer to Section 2.8.1)
- Alternatively, the output voltage diagnostic measurement can be read using the `CreateTCChannel` API, which reads the output voltage and converts it to the thermocouple temperature equivalent.
(refer to Section 2.8.2)

Note that each analog input for diagnostic measurements is offset from the analog output channel by 8.

Guardian diagnostic channels and CJC channels map as follows:

- AO channel 0 use 0, 1, 2 to read current, voltage, and internal ADC temperature
- AO channel 1 use 8, 9, 10 to read current, voltage, and internal ADC temperature
- and so forth up to AO channel 7 using 56, 57, 58
- CJC measurements are read on 61, 62, 63

You can read all diagnostic measurements on all channels and the three CJC measurements with the following channel list:

```
0:2,8:10,16:18,24:26,32:34,40:42,48:50,56:58,61:63
```

If you are reading only output voltages, you can use the following channel list:

```
1,9,17,25,33,41,49,57
```

2.8.1 Reading Guardian Diagnostic Values & CJC

The following code shows how to read diagnostic current, voltage (*in volts*) and temperature on AO channels 0 through 3 and read all 3 CJC measurements for a TC-378:

```
// create a Guardian input session to read current, voltage, temperature,
//   and CJC
```

```
CUeiSession guardianSession;
guardianSession.CreateAIChannel(
              "pdna://192.168.100.2/Dev1/Ai0:2,8:10,16:18,24:26,61:63",
               -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[15];
```

```
// read one scan
```

```
aiReader.ReadSingleScan(values);
```

2.8.2 Reading Guardian Output Voltage as TC Temperature

The following code shows how to read a CJC value and then read the Guardian output voltage on AO channels 0 through 3, which are returned as a *thermocouple temperature* that is compensated by the CJC measurement:

First, if you do not know a static CJC constant value for your system, read the CJC value:

```
// create a Guardian input session to read CJC

CUeiSession guardianCJCSession;
guardianCJCSession.CreateAIChannel(
                "pdna://192.168.100.2/Dev1/Ai61:63",
               -10.0, 10.0,
               UeiAIChannelInputModeDifferential);
```

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

guardianCJCSession.ConfigureTimingForSimpleIO();

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

CUeiAnalogScaledReader aiReader(guardianCJCSession.GetDataStream());
double values[3];

// start the session and read one scan

guardianCJCSession.Start();

aiReader.ReadSingleScan(values);

// Stop the session before reconfiguring channels.
// CleanUp if you want to remove the previously configured channels

aiReader.Stop();
aiReader.CleanUp();
```

Next, create a diagnostic thermocouple input session to read the output voltage on AO channels 0 through 3 that is returned as TC temperatures.
Also note that we provide the CJC constant that was measured above:

```
// Create a Guardian input session to read diagnostic voltage output
//    that will be converted to a thermocouple temperature

CUeiSession guardianTCSession;
guardianTCSession.CreateTCChannel(
            "pdna://192.168.100.2/Dev1/Ai1,9,17,25",
            -100.0, 100.0, UeiThermocoupleTypeK,
            UEITemperatureScaleCelsius, UeiCJCTypeConstant, values[0],
            “”,UeiAIChannelInputModeDifferential);
```

Then, configure timing and the reader.

```
// configure the Guardian input session for simple timing
guardianTCSession.ConfigureTimingForSimpleIO();
// the buffer must be big enough to contain one value per channel
CUeiAnalogScaledReader TCReader(guardianTCSession.GetDataStream());
double TCvalues[4];
// start the session and read one scan
guardianTCSession.Start();

TCReader.ReadSingleScan(TCvalues);
```

2.9 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();
  tcSimSession.CleanUp();
```

Chapter 3 Programming with the Low-level API

This chapter provides the following information about programming the TC-378 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 TC-378 (Immediate Mode) (Section 3.4)
  - Configuring Output Channels (Section 3.4.1)
  - Writing Output Data (Section 3.4.2)
  - Configuring Guardian Diagnostics & CJC (Section 3.4.3)
  - Reading Guardian Diagnostic & CJC Values (Section 3.4.4)
  - Reading Circuit Breaker Status & Reengaging (Section 3.4.5)
  - Advanced Configuration of Circuit Breakers (Section 3.4.6)

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 TC-378-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-TC-378

Function	Description
`DqAdv3xxWrite`	Write either floating point or raw values to TC-378 output.
`DqAdv378ReadADC`	Read back voltage, current, temperature from Guardian diagnostic ADCs.
`DqAdv378ADCEnable`	Enables or disables Guardian ADC. Note that the Guardian ADC is enabled when entering CONFIG mode (after power-up).
`DqAdv318Reengage`	Reset tripped circuit breakers for selected channels.
`DqAdv318CBStatus`	Get the circuit breaker status for selected channels.
`DqAdv318SetCBLevels`	Configure circuit breaker minimum and maximum trip levels for selected channels.
`DqAdv318SetConfig`	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 TC-378. 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, `Sample318` contains sample code for running the an TC-378 (and the AO-318 products) in Immediate mode and reading Guardian diagnostics.

3.3.1 Data Transfer Modes

The TC-378 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/RtVMap: 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
- RtVMap delivers multiple samples 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 TC-378 (Immediate Mode)

The following sections provide an overview of how to set up and use your TC-378 in Immediate Mode using the low-level API.

For best results, use this overview in conjunction with actual sample code, (e.g, Sample318). This overview does not address all initialization or error handling. Refer to Section 3.3 for sample code location.

3.4.1 Configuring Output Channels

Users initialize a list of TC-378 channels to enable and output samples to.

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

You can 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 Data

In Immediate mode, use the `DqAdv3xxWrite()` API to write raw or floating point samples to each of the enabled channels (in the order of the channel list):

```
uint16 data[CHANNELS];
double fdata[CHANNELS];

while (!stop) {

    //set up data to be output
    for (i = 0; i < CHANNELS; i++) {
          data[i] = 0; // not sending raw data
          fdata[i] = nextValueToOutput();   // +/- 125mV for TC-378
    }

    DqAdv3xxWrite(hd,        // handle to IOM
                  DEVN,      // position TC-378 inserted in the chassis
                  CHANNELS,  // total number of channels enabled
                  cl,        // channel list configured in previous step
                  0,         // RawData flag (0 is FALSE, use floating pt)
                  data,      // array of raw data - not used if RawData==0
                  fdata);    // CHANNELS-size array of floating point data

    UeiPalSleep(500); // controls data output rate

}
```

3.4.3 Configuring Guardian Diagnostics & CJC

UEI provides access to reading diagnostic Guardian current, voltage, and temperature, as well as up to 3 CJC measurements.

When accessing Guardian features, users configure which of the TC-378 diagnostic points to monitor for each output channel.

Diagnostic current, voltage, and temperature are defined as follows:

Diagnostic Channel	**TC-378 Description** (see **Figure 3-1**)
(0)	I_{in} current
(1)	AOut voltage at the connector pin
(2)	Temperature

Table 3-2. TC-378 Diagnostic Measurements

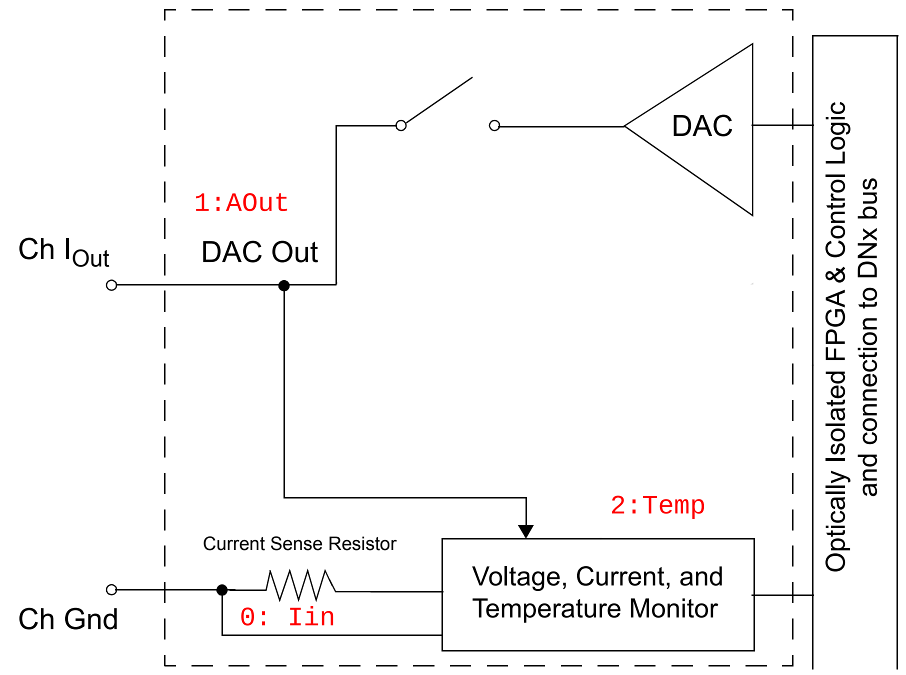
DAC
1:AOut
Ch I_Out
DAC Out
2:Temp
Current Sense Resistor
Ch Gnd
0: Iin
Voltage, Current, and Temperature Monitor
Optically Isolated FPGA & Control Logic and connection to DNx bus

Figure 3-1 Block Diagram of Diagnostic Measure Points on an TC-378 Channel

The 3 CJC measurements are specified with the following definitions:

- `DQ_TC378_CH_DCJC0` // channel number for TC-378 CJC #0
- `DQ_TC378_CH_DCJC1` // channel number for TC-378 CJC #1
- `DQ_TC378_CH_DCJC2` // channel number for TC-378 CJC #2

To configure Guardian and CJC measurements, users set up a read channel list with an array size of the total number of diagnostic data values that will be read:

Define the number of Guardian/CJC/Timestamp channels to read:

```
// RCHANNELS represents all diagnostic/CJC input channels to read:
//  Guardian current, voltage, & temperature, CJC, and optional timestamp

//  In this example, the 3 Guardian channels (diagnostic data points)
//  are configured for each of the enabled output channels (CHANNELS)
//  and we are configuring 3 CJC measurement channels

#define CHANNELS_CJC (3)           // number of CJC channels
#define TIMESTAMP    (0)           // no timestamp included for this example
#define RCHANNELS    ((3 * CHANNELS) + CHANNELS_CJC + TIMESTAMP)
```

Declare a read array for Guardian/CJC/Timestamp channel list:

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

Create Guardian/CJC/Timestamp channel list entries:

Create Guardian diagnostic channels entries (for measurements listed in Table 3-2) by logically ORing the output channel number using the UEI macro, `DQ_AO318_MAKE_CL(OutputChannel,DiagnosticChannel)`:

```
for (i = 0; i < CHANNELS; i++) {
    rcl[i*3+0] = DQ_AO318_MAKE_CL(i,0);     // read diagnostic Iin
    rcl[i*3+1] = DQ_AO318_MAKE_CL(i,1);     // read diagnostic AOut
    rcl[i*3+2] = DQ_AO318_MAKE_CL(i,2);     // read diagnostic Temp
}
```

Create CJC channels after the Guardian diagnostic channels are created:

```
if (CHANNELS_CJC == 3) {
    rcl[RCHANNELS-3-TIMESTAMP] = DQ_TC378_CH_DCJC0;
    rcl[RCHANNELS-2-TIMESTAMP] = DQ_TC378_CH_DCJC1;
    rcl[RCHANNELS-1-TIMESTAMP] = DQ_TC378_CH_DCJC2;
}
```

If configuring a timestamp reading, create that entry last or first:

```
if (TIMESTAMP) rcl[RCHANNELS - 1] = DQ_LNCL_TIMESTAMP;
```

Configure ADC hardware:

```
// 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

DqAdv378ReadADC(hd,         // handle to IOM
                DEVN,       // position TC-378 inserted in the chassis
                RCHANNELS, // total number of input (diagnostic/CJC channels)
                rcl,        // read channel list configured in previous step
                0,          // set to 0 for config
                0);         // set to 0 for config
```

3.4.4 Reading Guardian Diagnostic & CJC Values

The first call to the `DqAdv378ReadADC()` API is to program the ADC hardware, and then subsequent `DqAdv378ReadADC()` calls read the diagnostic data values (in the order of the `rcl` read channel list):

Declare arrays to hold returned measurements:

```
uint32 b_adcdata[CHANNELS*3 + CHANNELS_CJC + TIMESTAMP];
double f_adcdata[CHANNELS*3 + CHANNELS_CJC + TIMESTAMP];
```

Read Guardian diagnostic and CJC measurements (& optional timestamp):

```
DqAdv378ReadADC(hd,        // handle to IOM
                DEVN,      // position TC-378 inserted in the chassis
                RCHANNELS, // total number of diagnostic/CJC channels enabled
                rcl,       // read channel list
                b_adcdata, // array for returned raw readback data
                f_adcdata);// array for returned floating pt readback
```

3.4.5 Reading Circuit Breaker Status & Reengaging

To monitor the status of the circuit breakers, use the `DqAdv318CBStatus()` API, and to reengage circuit breakers, use the `DqAdv318Reengage` API.

Declare an array to hold circuit breaker status bits:

```
// declare an array to hold status states for board and all channels
//   cbstatus[0] is bitmask of all channels; cbstatus[1:8] are individual ch
uint32 cbstatus[DQ_AO318_CHAN+1]; // DQ_AO318_CHAN is all 8 channels
```

Read circuit breaker status:

```
// The channel mask is set as 0x1 is 1st channel; 0xff is all 8 ch
DqAdv318CBStatus(hd, DEVN, mask, 0, 0, cbstatus);
```

- `cbstatus[0]`:
  bit 23:16 - current status of CB for ch 7:0 (1 is tripped)
  (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 DqAdv318Reengage API a bitmask of channels to reengage
uint32 resetAllChannels = 0xff;

if ((yourCriteria == TRUE) && ((cbstatus[0]>>16) & 0xff)){
    DqAdv318Reengage(hd, DEVN, resetAllChannels);
}
```

3.4.6 Advanced Configuration of Circuit Breakers

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

The minimum and maximum defaults are set to the following:

- Current limits are ±8 mA
- ADC internal temperature limits are -45°C to 105°C

Advanced configuration is available to programmers using the low-level API. The circuit breaker has the following user programmable features:

- The circuit breaker can be enabled, disabled or controlled by up to 2 sources that are user programmable.
- You can select 0, 1, or 2 of the 3 diagnostic channels (see **Table 3-2**) as a control source to monitor for a circuit breaker. Selections are the current, voltage, and temperature Guardian measurements.
- Users can program a minimum or maximum value for any of the diagnostic channels selected for use as a control condition (or the min/max rule can be disabled to select 0).

In the following example, we configure the circuit breaker with only 1 control source. For that control source, we program a minimum and maximum trip point for the measurement:

1. Program circuit breaker on output voltage on channel 0
2. Reconfigure the trip points to be when the output voltage is less than a -100.0 mV minimum and greater than a 100.0 mV maximum. (Diagnostic channel (1) maps to the output voltage). (see **Figure 3-1**)

NOTE: The following example uses many predefined #defined variables to configure functionality. The variables used are explained in the example. To learn more, please refer to the PowerDNA API Reference Manual.

3.4.6.1 Declare CB Configuration Structures

To configure the circuit breaker, first declare Circuit Breaker configuration structures.

```
DQAO318CFG cfg318; // used by DqAdv318SetConfig()
DQAO318BRK_CFG cb_cfg; // used by DqAdv318SetCBLevels()
```

3.4.6.2 Program CB Enable & Criteria Selection

Set the `DQAO318CFG` structure to enable the circuit breaker and to use the output voltage as the control criteria.

Set mask to allow programming of the enable, source mode, and ADC channel list:

```
cfg318.prmmask = DQ_AO318_CBCFG_DAC |
                DQ_AO318_CBCFG_SETCBSRC |
                DQ_AO318_CBCFG_SETADCCL;
```

Enable the DAC:

```
cfg318.en_DAC   =   DQAO318CFG_DACEN_A; // enable CB_A
```

Select a source as the control:

```
cfg318.CB_mode  =  DQ_AO318_CB_A_SEL0(DQ_AO318_MINMAX_1) |
                    DQ_AO318_CB_A_SEL1(DQ_AO318_MINMAX_DISABLE)|
                    DQ_AO318_CB_B_SEL0(DQ_AO318_MINMAX_DISABLE)|
                    DQ_AO318_CB_B_SEL1(DQ_AO318_MINMAX_DISABLE)|
                   0;
```

NOTE: Users can choose to auto reengage the circuit breakers by ORing in a `DQ_AO318_CB_A_AUTO` flag into the `CB_mode` parameter.

Map the ADC diagnostic channels to `ADC_CL[]`, the circuit breaker ADC channel list.

The example in this section sets `CB_mode` to monitor `ADC_CL[1]`, `DQ_AO318_MINMAX_1`.

This causes the diagnostic ADC channel measurement that is programmed as the `ADC_CL[1]` element to be selected as the min/max control criteria.

(If we had programmed `DQ_AO318_CB_A_SEL0` with `DQ_AO318_MINMAX_0`, then `ADC_CL[0]` (I_{in}) could be the control criteria).

```
cfg318.ADC_CL[0] = DQ_AO318_ADC_CH_I;      //DQ_AO318_MINMAX_0 is Iin
cfg318.ADC_CL[1] = DQ_AO318_ADC_CH_AB_ext; // set ADC_CL[1] to AOut,
                                           // DQ_AO318_MINMAX_1 measurement
cfg318.ADC_CL[2] = DQ_AO318_ADC_CH_TEMP;   // DQ_AO318_MINMAX_2 is
                                          //                  temperature
```

Optionally, set the ADC sample rate and/or read count (which controls the number of readings outside the min/max that must occur before the circuit breaker trips):

```
cfg318.ADC_rate = 10.0;  // do not sample > 50Hz; recommended 10Hz or less
cfg318.rdcnt = 5;        // 0 to 15 (0 corresponds to 1 reading)
```

Call the `DqAdv318SetConfig` API to set configuration:

```
int mask = 0x1;   // channel mask; 0x1 is 1st channel (ch0); 0xff is all 8 ch
DqAdv318SetConfig(hd, DEVN, mask, 0, &cfg318);
```

3.4.6.3 Program CB Minimum & Maximum Levels

Set the `DQAO318BRK_CFG` structure to program the minimum and maximum levels for the diagnostic channels you programmed in `ADC_CL[0]`, `ADC_CL[1]`, and `ADC_CL[2]`.

NOTE: Note that what you select for `CB_mode` is what ADC diagnostic channel will actually be read and compared to its min/max as the criteria for tripping the circuit breaker. In our example, only `ADC_CL[1]` is selected in `CB_mode`; the other two (I_{in} and Temperature) will not be criteria.

Program min/max values:

```
cb_cfg.CB_val_min_f[0] = (float)-0.011;     // A: ADC_CL[0] is Iin, not used
cb_cfg.CB_val_max_f[0] = (float) 0.011;     // A
cb_cfg.CB_val_min_f[1] = (float)-0.10;      // min V: ADC_CL[1] is criteria
cb_cfg.CB_val_max_f[1] = (float) 0.10;      // max V
cb_cfg.CB_val_min_f[2] = (float)-45.0;      // deg C: ADC_CL[2] not used
cb_cfg.CB_val_max_f[2] = (float) 105.0;     // deg C
```

Program units:

```
cb_cfg.units[0] = 'I';      // this is 'I' because cfg318.ADC_CL[0] is set
                            //     to read the current Iin as ADC input
cb_cfg.units[1] = 'V';      // this is 'V' because cfg318.ADC_CL[1] is set
                            //     to read AOut voltage as ADC input
cb_cfg.units[2] = 'T';      // this is 'V' because cfg318.ADC_CL[2] is set
                            //     to read the temperature as ADC input
```

Call the `DqAdv318SetCBLevels` API to set min/max levels:

```
// channel mask: 0x1 is 1st channel (ch0); 0xff is all 8 ch
mask=0x1;

DqAdv318SetCBLevels(hd, DEVN, mask, &cb_cfg);
```

Appendix A

A.1 Accessories

The following cables and STP boards are available for the TC-378 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 x1.0h 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.

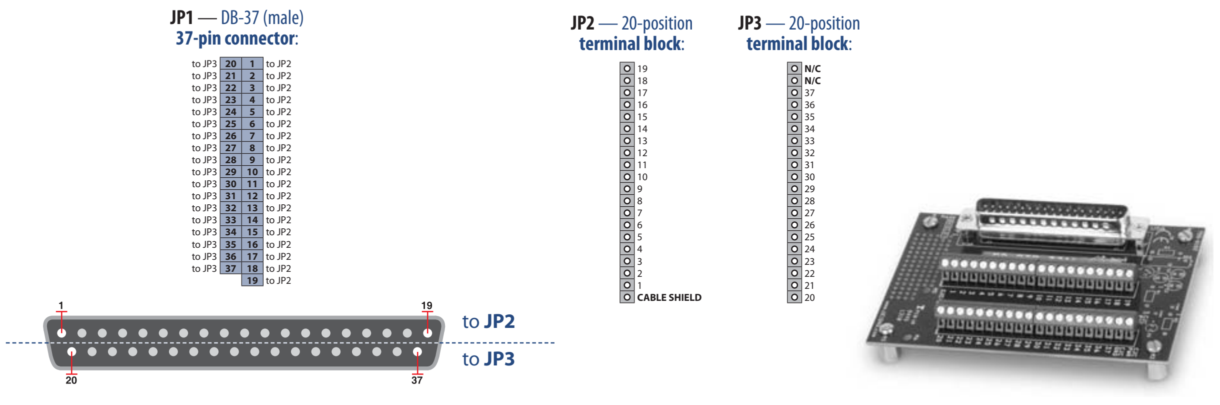

Figure A-1 Pinout and photo of DNA-STP-37 screw terminal panel

DNA-STP-378

The DNA-STP-378 is a Screw Terminal Panel with 3 built-in cold-junction compensation (CJC) temperature sensors.

Index