United Electronic Industries

The High-Performance Alternative

DNA/DNR-AI-225
Analog Input Layer
—
User Manual

Simultaneous Sampling, 24-bit, 25-channel
Analog Input Layer
for the PowerDNA Cube and RACKtangle chassis

August 2009 Edition
Version 3.6
PN Man-DNx-AI-225-0809

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

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

Contacting United Electronic Industries

Mailing Address:

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

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

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

Support:

Telephone:	(508) 9214600
Fax:	(508) 668-2350

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

Internet Support:

Support	support@ueidaq.com
Web-Site	www.ueidaq.com
FTP Site	ftp://ftp.ueidaq.com

Product Disclaimer:

WARNING!

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

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

Table of Contents

Table of Figures

Chapter 1 Introduction

This document outlines the feature-set and describes the operation of the DNx-AI-225-Analog Input boards. The DNA- version is designed for use with a PowerDNA Cube data acquisition system. The DNR- version is designed for use with a DNR-12 RACKtangle or a DNR-6 HalfRACK system. Both versions are functionally identical.

1.1 Organization of this Manual

This PowerDNx AI-225 User Manual is organized as follows:

- **Introduction**
  This chapter provides an overview of PowerDNA Analog Input Series board features, the various models available and what you need to get started.
- **The AI-225 layer**
  This chapter provides an overview of the device architecture, connectivity, and logic of the AI-225 layer.
- **Programming with the High-Level API**
  This chapter provides an overview of the how to create a session, configure the session for analog input, and interpret results on the AI-225 series layer.
- **Programming with the Low-Level API**
  This chapter describes low-level API commands for configuring and using the AI-225 series layer.
- **Appendices**
  This appendix provides a list of accessories available for AI-225 layer(s) and a description of protection circiuts.
- **Index**
  This is an alphabetical listing of the topics covered in this manual.

Conventions

To help you get the most out of this manual and our products, 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.

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**.”

1.2 AI-225 Layer Features

The AI-225 layer has the following features:

- 25 A/D simultaneously sampling converters with differential inputs
- 24-bit resolution, ±1.25V input range
    - Bipolar range ±1.25V, AIn– and AIn+ within –0.25V–5.0V
    - Unipolar range –0.25V..1.25V, AIn– connected to AGND directly or via up to 10KOhm resistor
    - Input underrange/overrange over the ±1.25V range is detected by the software and reported
- 5 S/s to 1000 S/s per channel sampling rates
- >120dB rejection of AC component of 50/60Hz for sampling rates below 10Hz
- >120dB Power Supply rejection ratio
- Typical 120dB of common mode rejection
- 120Hz –3dB analog front-end bandwidth
- ±15V overvoltage and 2kV ESD protection at every input
- One channel may be used for Cold Junction Compensation (CJC)
- Entire analog front end isolated from digital circuitry
- Direct Inputs for thermocouples
- May be used with RTDs, may be used with external excitation source (voltage excitation is available on the STP-AI-U)
- Strain gauge (bridge completion resistors required)
- Input Impedance 100 Mohm
- Input bias current ±15nA
- Non-linearity 3ppm (0.0003%)
- System noise 0.5µV RMS at 5Hz acquisition rate
- Effective number of bits: 23.5 @ 5Hz down to 19 @ 1kHz
- Input ground to system ground isolation: $350V_{rms}$
- Power consumption 3.9W max
- SYNC interface option (allows external triggering)

1.3 Device Architecture

Figure 1-1 is a block diagram of the architecture of the AI-225 layer.

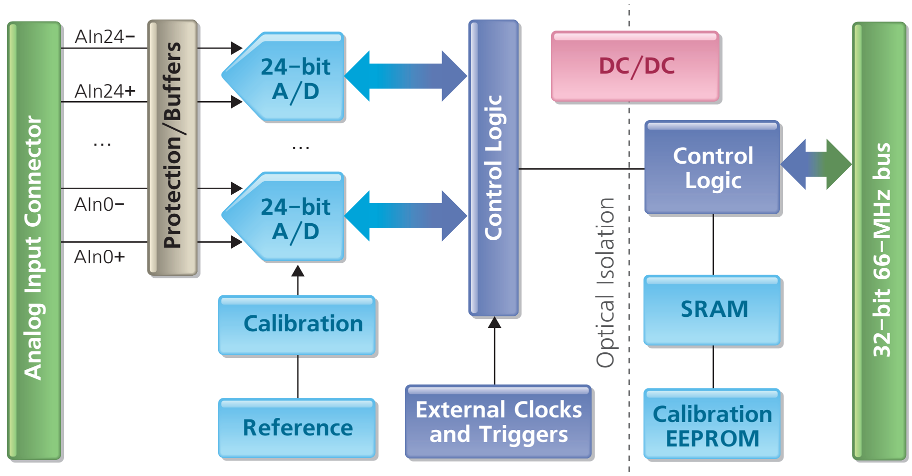


Figure 1-1 Block Diagram of the DNx-AI-225 Layer

As shown in **Figure 1-1**, the AI-225 layer has an independent converter for each of the 25 channels. A differential input signal goes first to an auto-zero buffer/ amplifier with 125Hz –3dB bandwidth and then to an A/D converter. The A/D converter accepts signals within a –0.25 to 5V range and measures up to a ±1.25V difference between AIN+ and AIn–. This mode is called **referenced bipolar differential** because both AIn+ and AIn- are referenced to system ground.

The AI-225 uses sigma-delta A/Ds that sample analog signals at 1.8MHz with a high over-sampling ratio and pass this data into decimating FIR filters. The over-sampling ratio varies from 64 to 32768 for various sampling rates. This inter-working of A/D converters allows the AI-225 to deliver true 24-bit resolution at a 5Hz data output rate down to 19-bit resolution at a 1 kHz rate.

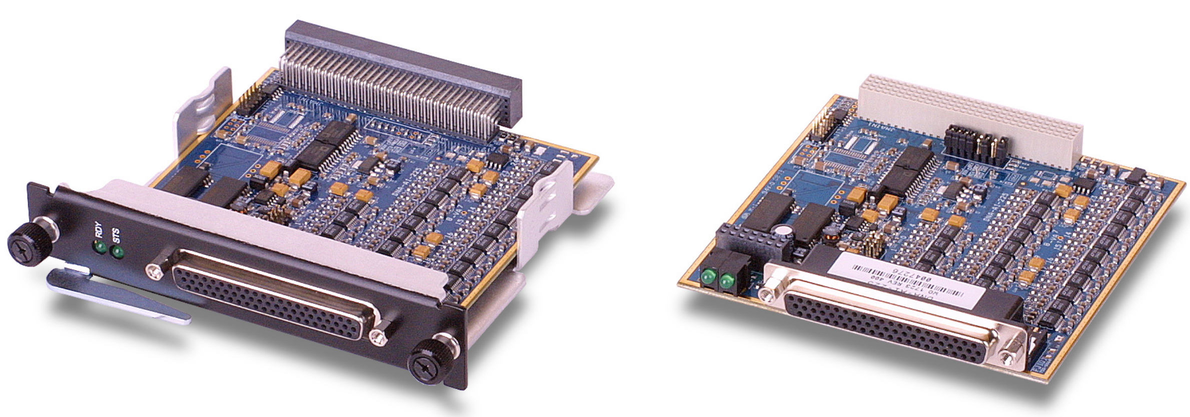


Figure 1-2. Photos of DNR- and DNA-AI-225 Analog Input Layer Boards

1.4 Layer Connectors and Wiring

The DNA-AI-225 layer supports referenced differential inputs only. Both signal and return line of the differential signal pair must have a potential within the range –0.25V to 5V, relative to isolated ground (AGND) level. **Figure 1-3** illustrates the pinout of the AI-225.

DB-62 (female)
62-pin connector:

62	DIO0	42	AGND	21	9V@20mA
61	AIN1	41	AIN0 Return	20	AIN0
60	AIN2 Return	40	AIN2	19	AIN1 Return
59	AIN4	39	AIN3 Return	18	AIN3
58	AIN5 Return	38	AIN5	17	AIN4 Return
57	AIN7	37	AIN6 Return	16	AIN6
56	AIN8 Return	36	AIN8	15	AIN7 Return
55	AIN10	35	AIN9 Return	14	AIN9
54	AIN11 Return	34	AIN11	13	AIN10 Return
53	AIN13	33	AIN12 Return	12	AIN12
52	AIN14 Return	32	AIN14	11	AIN13 Return
51	AIN16	31	AIN15 Return	10	AIN15
50	AIN17 Return	30	AIN17	9	AIN16 Return
49	AIN19	29	AIN18 Return	8	AIN18
48	AIN20 Return	28	AIN20	7	AIN19 Return
47	AIN22	27	AIN21 Return	6	AIN21
46	AIN23 Return	26	AIN23	5	AIN22 Return
45	AGND	25	AIN24 Return	4	AIN24
44	DIO2	24	AGND	3	9V@20mA
43	RESERVED	23	RESERVED	2	DIO1
		22	RESERVED	1	RESERVED

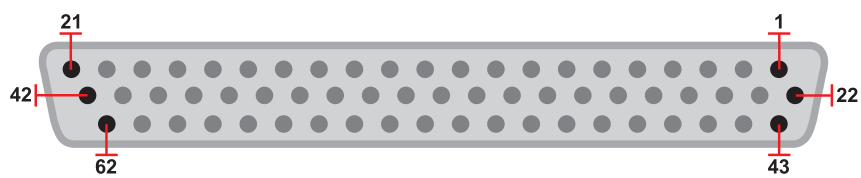


Figure 1-3. Pinout Diagram of the AI-225 Layer

1.4.1 Analog Input Ground Connections

To avoid errors caused by common mode voltages on analog inputs, follow the recommended grounding guidelines in **Figure 1-4** below.

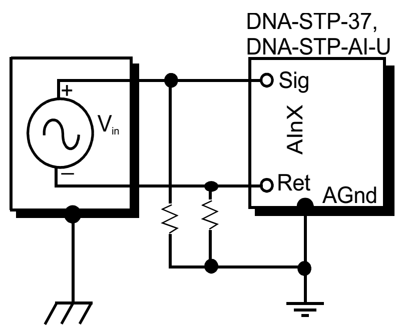
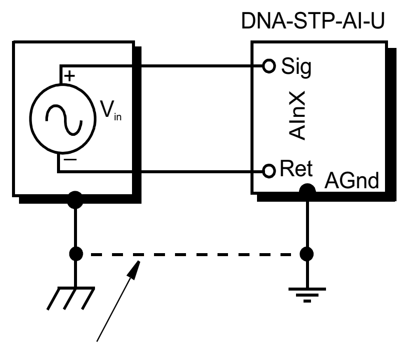
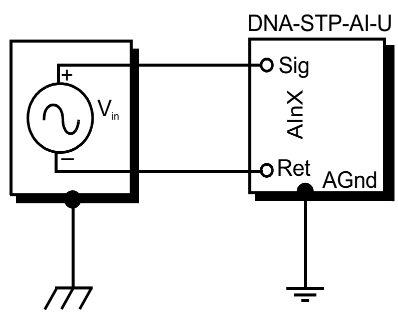
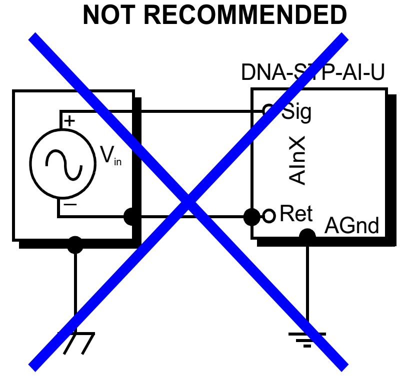
Input Configuration	**Type of Input**	
	Floating	**Grounded**
	Typical Signal Sources: **Thermocouples** **DC Voltage Sources** **Instruments or sensors with isolated outputs**	**Typical Signal Sources:** **Instruments or sensors with non-isolated outputs**
Differential	 Two resistors (10k <R< 100k) provide return paths to ground for bias currents.	 Add this connection to ensure that both grounds are at the same potential.
Single-Ended, Ground Referenced		**NOT RECOMMENDED** 

Figure 1-4. Recommended Ground Connections for Analog Inputs

Because all analog input channels in AI-201/202/207/208/225 layers are isolated as a group, you can connect layer AGND to the ground of the signal source and eliminate the resistors shown in **Figure 1-4** for floating differential input signals.

For RTD and Bridge Wiring, refer to **Figure 1-5**.

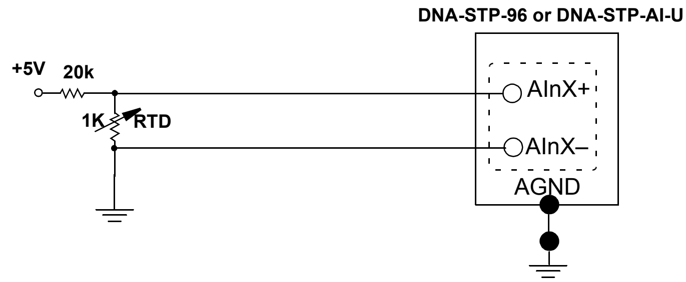


Figure 1-5. Analog Input Configuration — RTD and Bridge Wiring

NOTE: For more detail on RTD and thermocouple measurement connections, refer to the User Manual for the STP-AI-U Screw Terminal Panel, which may be downloaded at www.ueidaq.com.

1.5 Layer Capabilities

The AI-225 layer is capable of acquiring analog input voltages in ±1.25V range with gains of 1 at up to 24 bits of dynamic resolution (222nV RMS resolution).

The layer is capable of generating its own CL (channel list) clock and trigger, and deriving them from either local external lines from its connector or from the SYNCx bus.

Table 1-1. Gains

Card	Gain	Range	Noise, LSB	Resolution. noise limited
AI-225	1	±V1.25	1.5‘	222nV

The analog RC anti-aliasing filtering is tuned to provide roll-off at 1.5kHz (half of the maximum sampling frequency) as shown in **Table 1-2** below.

Table 1-2. Anti-Aliasing Filter Parameters

Frequency	Oversampling Ratio	Noise		ENOBs	
		ADC	System	ADC	System
1000	128	3.5µV	5µV	20	18
800	256	2µV	3µV	21.3	21
400	512	1.4µV	2µV	21.8	21
200	1024	1µV	1.5µV	22.4	22
100	2048	750nV	1µV	22.9	22
50	4096	510nV	900nV	23.4	23
25	8192	375nV	700nV	24	24
10	16384	250nV	600nV	24.4	24

1.5.1 Thermocouple Measurement

The AI-225 is capable of performing thermocouple measurements within 0.02°C at 10Hz per channel. The higher the speed of measurement, the more noise can be expected.

The following table shows test results for noise for the AI-225 when used in conjunction with the STP-AI-U terminal (10Hz/ channel acquisition speed):

Thermocouple Type	Temperature Range	P-p Noise, °C	100 Points RMS Noise, °C
B	Full Range of Thermocouples	0.50	0.16
C		0.24	0.07
E		0.06	0.02
J		0.07	0.02
K		0.10	0.03
N		0.12	0.03
R		0.32	0.1
S		0.37	0.1

Additional factors:

- Open TC detection circuitry on the DNA-STP-AI-U adds ~15µV (±2µV) constant offset on all channels. (May be compensated using the CJC temperature sensor calibration. Channel-channel difference may be adjusted using the offset calibration `simod 3` command.)
- CJC sensor is calibrated to better than 0.2°C accuracy at room temperature
- Stays within 0.4°C accuracy from –20 to +75°C temperature

1.6 Data Representation

The AI-225 layer is equipped with 25 24-bit A/D converters. The layer can return 24-bit two's complement data in 32-bit words, combined with levels on general-purpose digital I/O lines.

By default, if acquisition is not running, the output buffer is filled with relative addresses. On reset, every entry in the output buffer is filled with its relative position number. As an initializing step, you should read the buffer and discard the data before proceeding with normal data collection.

If you start receiving consecutive data from the layer (such as 0,1,2,..) it means that either the layer is not initialized properly or it is damaged.

The following definition converts raw data from the converter into a 24-bit straight binary value:

```
#define LT2440_GETVAL(V) (((V>>5)&0xffffff)^0x800000)
```

To convert data into floating point, use the following formula (V is a result from the `DQ_LT2440_GETVAL()` macro):

```
Volts = (V) * (2.5V/2^24) – 1.25V
```

Raw 32-bit data received from converter is represented as:

Bit	Name	Description	Reset State
31	**EOC**	Zero, if conversion is completed.	0
30	**DMY**	Always low.	0
29	**SIG**	Sign bit of the conversion. If V_{IN} is > 0, this bit is HIGH. If V_{IN} is < 0, this bit is LOW.	NA
28	**MSB**	Most significant bit of the result. If both Bit 29 and Bit 28 are HIGH, the differential input voltage is above +FS. If both Bit 29 and Bit 28 are LOW, the differential input voltage is below –FS.	NA
27-5	**LSB**	Less significant bits of the result.	NA
5-3	**SubLSB**	Sub LSB of the result beyond 24-bit level. Can be used in averaging.	NA
1	**DIO2**	Level of DIO1 line (output).	0
1	**DIO1**	Level of DIO1 line (input).	0
0	**DIO0**	Level of DIO0 line (input).	0

1.6.1 Software Calibration of this Layer

Unlike most PowerDNA layers, the AI-225 relies on software calibration on the host side.

The idea of layer calibration is to read, average, and store readings from all channels at zero volts and then at the fixed level (1V is the default level).

By subtracting actual readings at zero volts from the ideal value (0x800000, straight binary), the software calculates calibration offsets.

The firmware then subtracts the offset-adjusting value from the average value read at 1V. The result represents an offset-compensated reading at 1V. By dividing actual calibration voltage by offset-compensated reading, the software calculates the actual channel gain.

When DQE opens the IOM, it automatically downloads calibration coefficients from each AI-225 in the PowerDNA cube by issuing `DQCMD_RDFIFO` commands with `FIFO_GET_CAL FIFO` channel.

In return, the firmware sends a `CALSET_225_` structure, which contains the gain calibration level and the averaged readings at zero volts and that level.

Then, the software stores offset-adjusting values directly and calculates actual gain for every channel, as follows:

```
cfvolt = (cvolt/ONEVOLTINNV);
for (all channels) gain[i] = cfvolt/pcval[i];
```

You need to perform data calibration only when you configure and use the AI-225 layer directly, without DQE running.

Chapter 2 Programming with the High-Level API

This section describes how to program the PowerDNA AI-225 using the UEIDAQ's Framework API.

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

Although the following section focuses on the C++ API, the concept is the same no matter what programming language you use.

Please refer to the "UEIDAQ Framework User Manual" for more information on using other programming languages.

2.1 Creating a Session

The Session object controls all operations on your PowerDNA device. The first task, therefore, is to create a session object, as follows:

```
CUeiSession session;
```

2.2 Configuring the Channels

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 the device class is **pdna**.

2.2.1 Voltage Measurement

To program the analog input circuitry, configure the channel list using the session's object method "CreateAIChannel".

For example, the following resource string selects analog input channels 0,2,3,4 on device 1 at IP address 192.168.100.2:

```
"pdna://192.168.100.2/Dev1/Ai0,2,3,4"
```

The session object's method "CreateAIChannel" is used to configure the channel list, gain, and signal referencing mode. The gain and input mode parameters are ignored when using an AI-225 because it doesn't have programmable gain and is differential only).

```
// Configure session to acquire from channels 0 and 1
session.CreateAIChannel("pdna://192.168.100.2/Dev0/Ai0,1",
                        -0.15, 0.15,
                        UeiAIChannelInputModeDifferential);
```

2.2.2 Thermocouple Measurement

The AI-225 offers very good accuracy, thanks to its 24-bit A/D converters and its dedicated channel for measuring the Cold Junction Compensation temperature sensor. Therefore, it is well suited for measuring temperature with thermocouples.

Use the session object's method "CreateTCChannel" to configure the channels, thermocouple type, CJC sensor, and temperature scale.

You can use thermocouples of type E, J, K, R, S, T, B or N.

You can measure the CJC temperature either by using a sensor integrated in the terminal block or by specifying a constant.

You can measure temperature in degrees Celsius, Fahrenheit, Kelvin, or Rankine.

```
// Configure the session to acquire temperatures in degrees
// Celsius from channels 0 and 1 using K thermocouples.
// Use the built-in CJC sensor
session.CreateTCChannel("pdna://192.168.100.2/Dev0/Ai0,1",
                        -100.0, 100.0,
                        UeiThermocoupleTypeK,
                        UeiTemperatureScaleCelsius,
                        UeiCJCTypeBuiltIn,
                        0.0, "",
                        UeiAIChannelInputModeDifferential);
```

2.2.3 RTD Measurement

RTD measurements are configured using the Session object method "CreateRTDChannel".

RTD sensors are resistive sensors whose resistance varies with temperature. Knowing the resistance of an RTD, we can calculate the temperature using the "Callendar Van-Dusen" equations.

RTD sensors are specified using the "alpha" (a) constant. It is also known as the temperature coefficient of resistance, which 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, which is used to convert resistance measurements to temperature.

To measure the RTD resistance, we need to know the amount of current flowing through it. We can then calculate the resistance by dividing the measured voltage by the known excitation current.

To measure the excitation current, we measure the voltage from a high precision reference resistor whose resistance is known.

The reference resistor is built-into the terminal block if you are using a DNA-STP-AI-U, but you can provide your own external reference resistor, if you prefer.

In addition, you must configure the RTD type and its nominal resistance at 0° Celsius, as shown in the following example.

```
// Add 4 channels (0 to 3) to the channel list and configure
// them to measure a temperature between 0.0 and 200.0 deg. C.
// The RTD sensor is connected to the DAQ device using
// two wires, the excitation voltage is 5V, and the reference
// resistor is the 20kOhms resistor built-into the DNA-STP-AI-U.
// The RTD alpha coefficient is 0.00385, the nominal resistance at 0°
C is
// 100 Ohms, and the measured temperature will be returned in degrees
// Celsius.

MySession.CreateRTDChannel("pdna://192.168.100.2/dev0/Ai0:3",
                           0, 1000.0,
                           UeiTwoWires,
```

```
5.0,
UeiRefResistorBuiltIn,
20000.0,
UeiRTDType3850,
100.0,
UeiTemperatureScaleCelsius,
UeiAIChannelInputModeDifferential);
```

2.3 Configuring the Timing

You can configure the AI-225 to run in simple mode (point by point) or buffered mode (ACB mode).

In simple mode, the delay between samples is determined by software on the host computer.

In buffered mode, the delay between samples is determined by the AI-225 on-board clock.

The following sample shows how to configure the simple mode. Please refer to the “UEIDAQ Framework User Manual” to learn how to use the other timing modes.

```
session.ConfigureTimingForSimpleIO();
```

2.4 Reading Data

Reading data from the AI-225 is done using a reader object. There is a reader object to read raw data coming straight from the A/D converter. There is also a reader object to read data already scaled to volts or temperatures.

The following sample code shows how to create a scaled reader object and read samples.

```
// Create a reader and link it to the session’s stream
CueiAnalogScaledReader reader(session.GetDataStream());

// read one scan, the buffer must be big enough to contain
// one value per channel
double data[2];
reader.ReadSingleScan(data);
```

2.5 Cleaning-up the Session

The session object cleans itself up when it goes out of scope or when it is destroyed. However, to reuse the object with a different set of channels or parameters, you can also clean up the session manually.

```
session.CleanUp();
```

Chapter 3 Programming with the Low-Level API

This section describes how to program the PowerDNA cube using the low-level API. The low-level API offers direct access to PowerDNA DAQBios protocol and also allows you to directly access device registers.

We recommend that you use the UEIDAQ Framework (see Chapter 2), because it is easier to use.

You should only need to use the low-level API if you are using an operating system other than Windows.

3.1 Configuration Settings

Configuration setting are passed in `DqCmdSetCfg()` and `DqAcbInitOps()` functions.

Note that not all configuration bits apply to AI-225 layer.

The following bits make sense:

```
#define DQ_FIFO_MODEFIFO (2L << 16) // continuous acquisition with
                                    //  FIFO
#define DQ_LN_MAPPED     (1L<<15)   // For WRRD (DMAP) devices
#define DQ_LN_STREAMING  (1L<<14)   // For RDFIFO devices - stream the
                                    // FIFO data automatically
                                    // For WRFIFO - do NOT send reply
                                    // to WRFIFO unless needed
#define DQ_LN_IRQEN      (1L<<10)   // enable layer irqs
#define DQ_LN_PTRIGEDGE1 (1L<<9)    // stop trigger edge MSB
#define DQ_LN_PTRIGEDGE0 (1L<<8)    // stop trigger edge:
                                    // 00 - software,
                                    // 01 - rising, 02 - falling
#define DQ_LN_STRIGEDGE1 (1L<<7)    // start trigger edge MSB
#define DQ_LN_STRIGEDGE0 (1L<<6)    // start trigger edge:
                                    // 00 -software, 01 - rising,
                                    // 02 - falling
#define DQ_LN_CVCKSRC1   (1L<<5)    // CV clock source MSB
#define DQ_LN_CVCKSRC0   (1L<<4)    // CV clock source 01 - SW, 10 -
HW, 11 -EXT
#define DQ_LN_CLCKSRC1   (1L<<3)    // CL clock source MSB
#define DQ_LN_CLCKSRC0   (1L<<2)    // CL clock source 01 - SW, 10 -
                                    // HW, 11 -EXT
#define DQ_LN_ACTIVE     (1L<<1)    // "STS" LED status
#define DQ_LN_ENABLED    (1L<<0)    // enable operations
```

For streaming operations with hardware clocking, select the following flags:

```
DQ_LN_ENABLE | DQ_LN_CVCKSRC0 | DQ_LN_STREAMING | DQ_LN_IRQEN |
DQ_LN_ACTIVE
```

`DQ_LN_ENABLE` enables all operations with the layer.

`DQ_LN_CVCKSRC0` selects the internal channel list clock (CL) source as a timebase. The AI-225 supports CV clock.

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

You can select either the CL or CV clock as a timebase. Because of the parallel architecture of AI-225 layer, either clock triggers all converters.

Aggregate rate = Per-channel rate * Number of channels

Acquisition rate cannot be selected on per-channel basis. To select a different resulting rate for a different channel, program the proper decimators in the FIR unit.

3.2 Channel List Settings

The AI-225 layer has a very simple channel list structure, as shown in the table below.:

Bit	Name	Purpose
31	`DQ_LNCL_NEXT`	Tells firmware there is a “next” entry in the channel list.
20	`DQ_LNCL_TSRQ`	Request timestamp as a next data point
7..0		Channel number

3.3 Layer-specific Commands and Parameters

Layer-specific functions are described in the DaqLibHL.h file.

DqAdv225Read()

This function works using underlying `DqReadAIChannel()`, but converts the data using internal knowledge of the input range and calibrates every channel. It uses `DQCMD_IOCTL` with `DQIOCTL_CVTCHNL` under the hood.

When this function is called for the first time, the firmware stops any ongoing operation on the device specified and reprograms it according to the channel list supplied. This function uses the preprogrammed CL update frequency — 13.75Hz. You can reprogram the update frequency by calling `DqCmdSetClk()` after the first call to `DqAdv225Read()`.

Therefore, you cannot perform this function call when the layer is involved in any streaming or data mapping operations.

If you specify a short timeout delay, this function can time out when called for the first time because it is executed as a pending command and layer programming takes up to 10ms.

Once this function is called, the layer continuously acquires data and every call to the function returns the latest acquired data.

If you want to cancel ongoing sampling, call the same function with 0xFFFFFFFF as a channel number.

3.4 Using the Layer in ACB Mode

The following is a pseudo-code example that highlights the functions needed in sequence to use ACB on the 225 layer. A complete example with error checking can be found in the directory SampleACB205.

```
#include "PDNA.h"
// unit configuration word
#define CFG225          (DQ_LN_ENABLED \
                        |DQ_LN_ACTIVE \
                        |DQ_LN_GETRAW \
                        |DQ_LN_IRQEN \
                        |DQ_LN_CLCKSRC0 \
                        |DQ_LN_STREAMING \
                        |DQ_AI225_MODEFIFO)
    uint32 Config = CFG225;
```

STEP 1: Start DQE engine.

```
#ifndef _WIN32
    DqInitDAQLib();
#endif

    // Start engine
    DqStartDQEngine(1000*1, &pDqe, NULL);

    // Open communication with IOM
    hd0 = DqOpenIOM(IOM_IPADDR0, DQ_UDP_DAQ_PORT, TIMEOUT_DELAY,
&RdCfg);

    // Receive IOM crucial identification data
    DqCmdEcho(hd0, DQRdCfg);

    // Set up channel list
    for (n = 0; n < CHANNELS; n++) {
        CL[n] = n;
    }
```

STEP 2: Create and initialize host and IOM sides.

```
    // Now we are going to test device
    DqAcbCreate(pDqe, hd0, DEVN, DQ_SS0IN, &bcb);

    // Let’s assume that we are dealing with AI-201 device
    dquser_initialize_acb_structure();

    // Now call the function
    ret = DqAcbInitOps(bcb,
                       &Config,
```

```
                        0,      //TrigSize,
                        NULL,   //pDQSETTRIG TrigMode,
                        &fCLClk,
                        0,      //float* fCVClk
                        &CLSize,
                        CL,
                        0,      //uint32* ScanBlock,
                        &acb);    printf("Actual clock rate: %f\n",
fCVClk);

    // Now set up events
    DqeSetEvent(bcb,
DQ_eFrameDone|DQ_ePacketLost|DQ_eBufferError|DQ_ePacketOOB);
```

STEP 3: Start operation.

```
    // Start operations
    DqeEnable(TRUE, &bcb, 1, FALSE);
```

STEP 4: Process data.

```
 // We will not use event notification at first - just retrieve scans
    while (keep_looping) {

       DqeWaitForEvent(&bcb, 1, FALSE, EVENT_TIMEOUT, &events);


        if (events & DQ_eFrameDone) {
            minrq = acb.framesize;
            avail = minrq;
            while (TRUE) {
          DqAcbGetScansCopy(bcb, data, acb.framesize, acb.framesize,

                 &size, &avail);
                samples += size*CHANNELS;

                for (i = 0; i < size * CHANNELS; i++) {
                    fprintf(fo, "%f\t", *((float*)data + i));
                    if ((i % CHANNELS) == (CHANNELS - 1)) {
                         fprintf(fo, "\n");
                    }
                }

 printf("eFD:%d scans received (%d samples) min=%d avail=%d\n", size,
                  samples, minrq, avail);
                if (avail < minrq) {
                    break;
                }
            }
```

```
        }
    }
```

STEP 5: Stop operation.

```
    DqeEnable(FALSE, &bcb, 1, FALSE);
```

STEP 6: Clean up.

```
    DqAcbDestroy(bcb);
    DqStopDQEngine(pDqe);
    DqCloseIOM(hd0);
#ifndef _WIN32
    DqCleanUpDAQLib();
#endif
```

3.5 Using Layer in DMap Mode

```
#include "PDNA.h"
```

STEP 1: Start DQE engine

```
#ifndef _WIN32
    DqInitDAQLib();
#endif

    // Start engine
    DqStartDQEngine(1000*10, &pDqe, NULL);

    // open communication with IOM
    hd0 = DqOpenIOM(IOM_IPADDR0, DQ_UDP_DAQ_PORT, TIMEOUT_DELAY,
&DQRdCfg);

    // Receive IOM crucial identification data
    DqCmdEcho(hd0, DQRdCfg);
    for (i = 0; i < DQ_MAXDEVN; i++) {
        if (DQRdCfg->devmod[i]) {
            printf("Model: %x Option: %x\n", DQRdCfg->devmod[i],
DQRdCfg->option[i]);
} else {
            break;
}
    }
```

STEP 2: Create and initialize host and IOM sides.

```
    DqDmapCreate(pDqe, hd0, &pBcb, UPDATE_PERIOD, &dmapin, &dmapout);
```

STEP 3: Add channels into DMap.

```
    for (i = 0; i < CHANNELS; i++) {
        DqDmapSetEntry(pBcb, DEVN, DQ_SS0IN, i, DQ_ACB_DATA_RAW, 1,
&ioffset[i]);
        printf("offset%d = 0x%x\n", i, (uint32)ioffset[i]);
    }

    DqDmapInitOps(pBcb);

    DqeSetEvent(pBcb,
DQ_eDataAvailable|DQ_ePacketLost|DQ_eBufferError|DQ_ePacketOOB);
```

STEP 4: Start operation.

```
    DqeEnable(TRUE, &pBcb, 1, FALSE);
```

STEP 5: Process data.

```
    while (keep_looping) {

        DqeWaitForEvent(&pBcb, 1, FALSE, timeout, &eventsin);

        if (eventsin & DQ_eDataAvailable) {
            printf("\ndata ");
            for (i = 0; i < CHANNELS; i++) {
                printf("%08x ", *(uint32*)ioffset[i]);
            }
        }
    }
```

STEP 6: Stop operation.

```
    DqeEnable(FALSE, &pBcb, 1, FALSE);
```

STEP 7: Clean up.

```
    DqDmapDestroy(pBcb);
    DqStopDQEngine(pDqe);
    DqCloseIOM(hd0);
#ifndef _WIN32
    DqCleanUpDAQLib();
#endif
```

Appendices

A - Accessories

The following cables and STP boards are available for the AI-225 layer.

DNA-CBL-62

2.5ft, 62-way round shielded cable

DNA-STP-AI-U

Universal PowerDNA Universal Analog Input Screw Terminal Panel

DNA-STP-62

62-channel screw terminal panel

DNA-5B-CONN

24-channel signal-conditioning mating panel

B – Protection Circuits

At times, the signal being measured appears to clip when attempting to measure what appear to be normal voltages. Clipping is a sign of tripping the protection circuitry of the layer, which is designed to protect layer components from damage by high voltage. This can happen under a variety of conditions, such as a different potential for the AI-225's ground *vs.* the chassis of the instrument (this can be fixed by connecting the chassis to the AGND line on the STP-AI-U).

Given the variables:

Let V_{EE} = 0V (this is AGND)

Let V_{CC} = 5V (or anywhere between 4.75V to 5.25V, depending on the P.S.U.)

Breaking the following rules activates the protection circuitry:

1. $V_{CC} + 0.25V > V_{IN}(-) > V_{EE} - 0.25V$

$V_{CC} + 0.25V > V_{IN}(+) > V_{EE} - 0.25V$

2. $V_{IN}(-) - V_{IN}(+) | \leq 1.25V$

Index