United Electronic Industries

The High-Performance Alternative

DNA-AO-332
Analog Output Layer
User Manual

March 2009 Edition
PN Man-DNA-AO-332-0309
Version 1.5

See the UEI website for complete terms and conditions of sale:

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

Contacting United Electronic Industries

Mailing Address:

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

For a list of our distributors and partners in the US and around the world, please see http://www.ueidaq.com/partners/

Support:

Telephone:	(508) 921-4600
Fax:	(508) 668-2350

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

Internet Support:

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

Product Disclaimer:

WARNING!

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

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

Table of Contents

List of Figures

Chapter 1 Introduction

This document outlines the feature set and use of the DNA-AO-332 32-channel, 16-bit, analog output layer when used with the UEI “Cube” or “*RACK*tangle” I/O chassis. The DNA-AO-332 is a high density, high precision, analog voltage output layer with an output range of $\pm$10 V that can drive $\pm$10 mA per channel. The board has 32 independent DACs, one for each channel, with per-channel digital offset and gain calibration that limits initial gain error to $\pm$450 μV and offset error to $\pm$300 μV.

All 32 channels may be configured to update either simultaneously or one at a time. A 1024-sample FIFO on each channel allows each DAC to be updated at 10k samples/sec without losing any data.

Double-buffered outputs combined with low-glitch DACs make the DNA-AO-332 board an ideal choice for producing low-frequency waveforms or for generating high-precision switched stimuli.

A digital input is included for use as a trigger or as a general purpose digital input. A single general purpose digital output is also provided.

Software for the board is provided as part of the UEIDAQ Framework package. This provides an easy-to-use API that supports all popular Windows programming languages and most real-time operating systems such as QNX, RTX, RT Linux, and others. The UEIDAQ Framework is also designed for use with LabVIEW, MATLAB/Simulink, DASYLab, and any ActiveX or OPC server application package.

1.1 Organization

This DNA AO-332 User Manual is organized as follows:

- **Introduction**
  This chapter provides an overview of DNA-AO-332 board/layer features, accessories, and what you need to get started.
- **DNA-AO-332 Layer**
  This chapter provides an overview of the device architecture, connectivity, logic, and accessories for the DNA-AO-332 layer board.
- **Programming with the High-Level API**
  This chapter provides a general overview of procedures that show how to create a session, configure the session, and generate output on a DNA-AO-332 layer, working with the UEIDAQ Framework High-Level API.
- **Programming with the Low-Level API**
  This chapter describes the Low-Level API commands for configuring and using a DNA-AO-332 layer.
- **Appendix A – Accessories**
  This appendix provides a list of accessories available for use with a DNA-AO-332 layer.
- **Index**
  This is an alphabetical index of topics covered in this manual.

NOTE: A glossary of terms used with the PowerDNA Cube and Layers can be viewed and/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 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**."

1.2 The DNA-AO-332 Analog Output Layer

Technical specifications for the DNA-AO-332 Analog Output Layer are listed in **Table 1-1**.

Table 1-1. DNA-AO-332 Technical Specifications

Technical Specifications:

Number of Channels	32
Resolution	16-bits
Max Update Rate:	10 kHz/channel (320 kHz max aggregate)
FIFO Buffer Size	1024 samples
Type of D/A	double-buffered
INL (no load)	±4 LSB (0.012%)
DNL (no load)	±1 LSB (0.003%)
Monotonicity	16 bits guaranteed over temperature
Gain Calibration Error	±450 µV, typ.
Offset Calibration Error	±306 µV, typ.
Offset Drift	5ppm/°C
Gain Drift	5ppm/°C
Output Range	±10 V
Output Coupling	DC
Output Impedance	0.1 Ω (typ)
Current Drive	±10 mA/channel
Capacitive Loads	500 pF
Settling Time	50 µs to 16 bits
Slew Rate	1 V/µs
Power up state	0 V ±10 mV
Digital I/O	1 digital input, 1 digital output (logic level)
Isolation	350Vrms
Power Consumption	2.0W - 3W (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*	50 g, 3 ms half sine, 18 shocks @ 6 orientations 30 g, 11 ms half sine, 18 shocks @ 6 orientations
MTBF	TBD hours

1.2.1 Features

The major features of the DNA-AO-332 Analog Output board are as follows:

- 32 independent DACs
- 16-bit resolution
- 10 kHz per channel maximum update rate
- ±10 V output range, ±10 mA per channel
- Low glitch output
- Per-channel offset and gain calibration
- Simultaneous update across all channels, if desired

Figure 1-1 is a photo of the DNA-AO-332 Layer board.

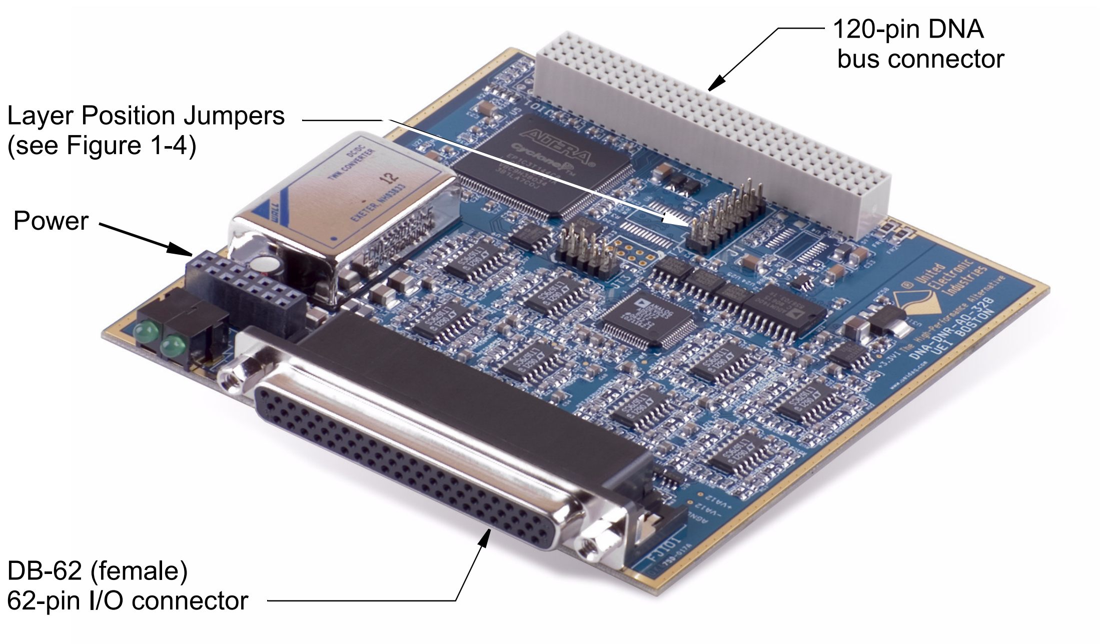

120-pin DNA bus connector
Layer Position Jumpers (see Figure 1-4)
Power
DB-62 (female) 62-pin I/O connector

Figure 1-1. DNA-AO-332 Board

1.3 Device Architecture

The DNA-AO-332 Analog Output Layer board has 32 individual analog output channels. A Block Diagram of the board/layer is shown in **Figure 1-2**.

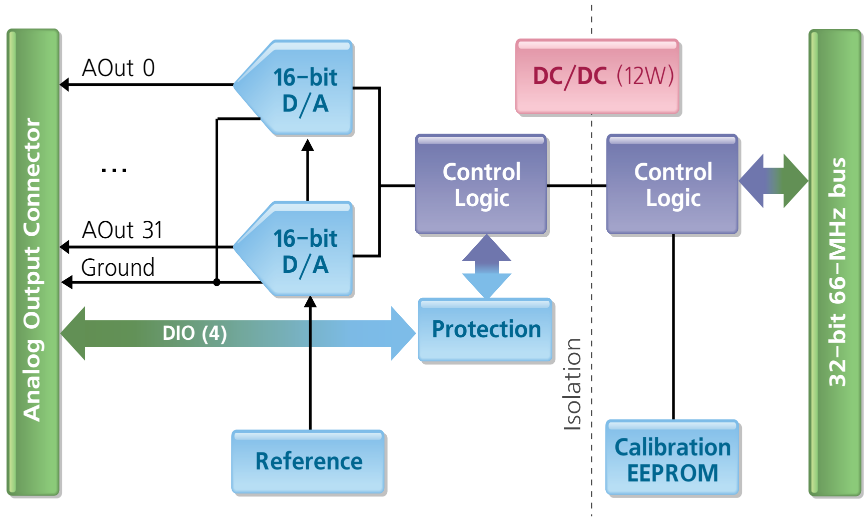


Figure 1-2. Block Diagram of DNA-AO-332 Device Architecture

As shown in the diagram above, the DNA-AO-332 board uses an independent 16-bit DAC for each channel. Offset and gain calibration data for each channel is stored in a calibration EEPROM, which automatically corrects each data sample as needed. A single digital input and a single digital output are also provided for general purpose use.

Note that complete electrical isolation is provided between the system bus/logic and the external I/O circuits.

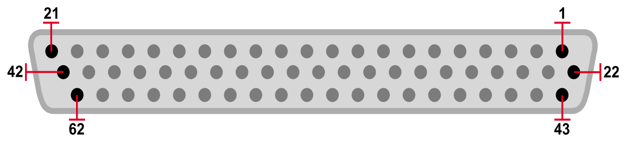
1.4 Layer Connectors and Wiring

Since the DNA-AO-332 Analog Output board is designed with output buffers, separate force and sense lines are not provided. .

1.4.1 Connectors

The pinout of the 62-pin connector for the DNA-AO-332 Layer board is shown in **Figure 1-3**. A physical layout of the board is shown in **Figure 1-4**.

.



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

Figure 1-3. DNA-AO-332 DB-62 I/O Connector Pinout

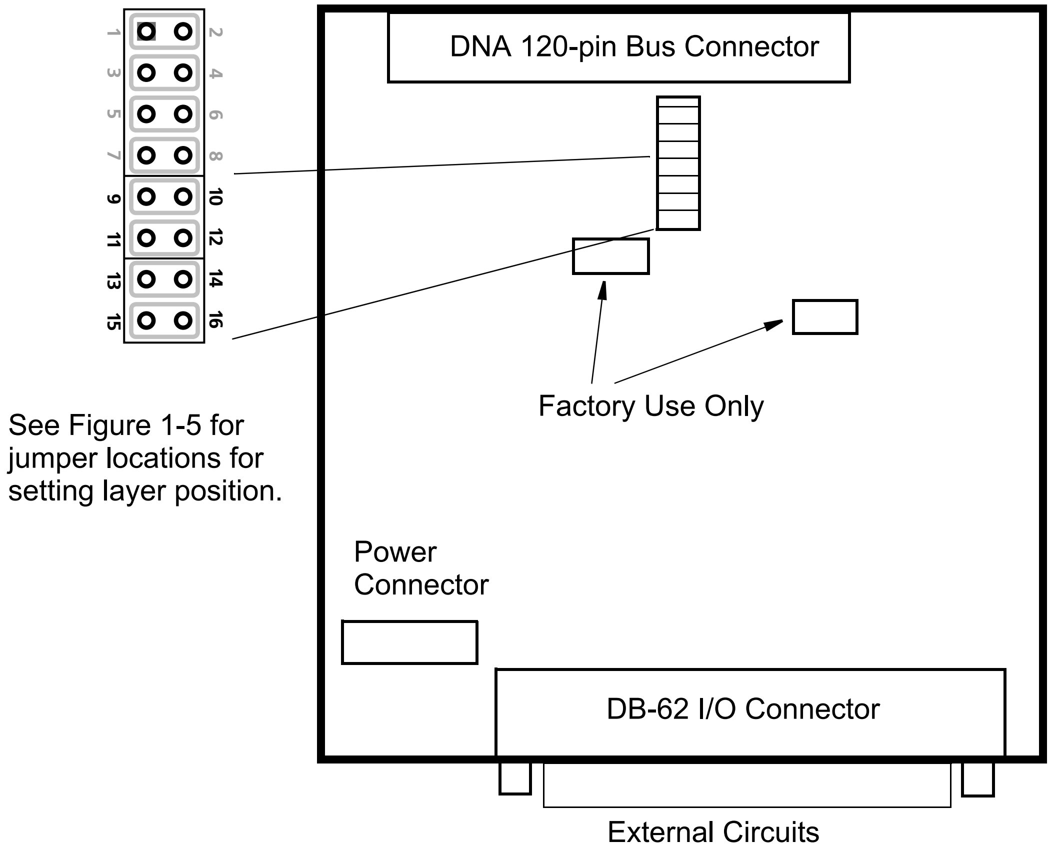


Figure 1-4. Physical Layout of DNA-AO-332 Layer Board

1.4.1.1 Jumper Settings

A diagram of the jumper block is shown in **Figure 1-4**. To set the layer position jumpers, place jumpers as shown in **Figure 1-5**.

		Layer's Position as marked on the Faceplate*					
		I/O **1**	I/O **2**	I/O **3**	I/O **4**	I/O **5**	I/O **6**
Jx Pins	9-10	●	○	●	○	●	○
	11-12	●	●	○	○	●	●
	13-14	●	●	●	●	○	○
	15-16	●	●	●	●	●	●

* *All I/O Layers are sequentially enumerated from top to the bottom of the Cube*
○ - **Open** ● - **Closed**

Figure 1-5. Diagram of DNA-AO-332 Layer Position Jumper Settings

Chapter 2 Programming with the High-Level API

2.1 Programming with the UEIDAQ Framework API

This section describes how to program the DNA-AO-332 board/layer using the UEIDAQ Framework High-Level API.

The UEIDAQ Framework is object-oriented. Its objects can be manipulated in the same manner within 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 for any programming language you use.

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

Please refer to the examples that come with the UEIDAQ Framework. They contain detailed and commented code that can be compiled and executed.

2.1.1 Creating a Session

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

```
CUeiSession session;
```

2.1.2 Configuring the Channels

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

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

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

For example, the following resource string selects analog output channels 0,1 on device 1 at IP address 192.168.100.2: “pdna://192.168.100.2/Dev1/Ao0:1”

```
// Configure channels 0,1 with an output
// range of ±10V
session.CreateAOChannel("pdna://192.168.100.2/
Dev0/ao0:1",-10.0, 10.0);
```

2.1.3 Configuring the Timing

You can configure the DNA-AO-332 layer to run either 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 DNA-AO-332 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();
```

2.1.4 Writing Data

Writing data to the DNA-AO-332 board/layers is done with a writer object. You can create a writer object that writes raw data straight to the D/A converter. You can also create a writer object that writes data scaled to volts. Framework automatically performs a conversion to binary code 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 reader and link it to the
// session’s stream
CueiAnalogScaledWriter
writer(session.GetDataStream());

// write one scan, the buffer must contain
// one value
// for each channel
double data[2] = {0.0, 0.0};
writer.WriteSingleScan(data);
```

Similarly, you can create a raw writer object by entering the following:

```
// Create a reader and link it to the session’s
stream
CUeiAnalogRawWriter
writer(session.GetDataStream());
// write one scan, the buffer must contain one
value
// for each channel
uInt16 data[2] = {0x1234, 0x5678};
writer.WriteSingleScan(data);
```

2.1.5 Cleaning-up the Session

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

```
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 access device registers directly.

We recommend that, where possible, you use the UEIDAQ Framework high-level API (*see* "Programming with the UEIDAQ Framework API" on page 7), which is easier to use than the low-level API.

You need to use the low-level API only if you are using an operating system other than Windows or if you are using a UEIPAC Programmable Automation Controller.

3.1 Configuration Settings

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

Not all configuration bits apply to DNA-AO-332 board, however. 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 (automatically
selected)
#define DQ_LN_STREAMING  (1L<<14)
     // For RDFIFO devices - stream the FIFO data
   //(automatically selected) 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_AO3xx_BI10
```

DNA-AO-332 has a range of layer-specific settings - as follows:

The following modes are reserved for future use:

```
#define DQ_AO3xx_MODEFIFO  (1L << 19)
     // continuous output with FIFO
#define DQ_AO3xx_MODECONT  (2L << 19)
    // waveform mode – continuous
#define DQ_AO3xx_MODECYCLE (xL << 19)
    // waveform mode – regenerate
#define DQ_AO3xx_MODEWFGEN (4L << 19)
   // waveform mode - hardware
```

`DQ_LN_ENABLE` enables all operations with the layer. `DQ_LN_CVCKSRC0` selects the internal channel list clock (CL) source as a timebase. AO-332 supports CV clock.
`DQ_LN_ACTIVE` is needed to switch on “STS” LED on the CPU layer.

You can select either the CL or CV clock as a timebase. Because of the parallel architecture of AO-332 layer, either clock triggers all converters.

```
Aggregate rate = Per-channel rate * Number of 
channels
```

Note that acquisition rate cannot be selected on per-channel basis.

3.2 Channel List Settings

The DNA-AO-332 layer has the following channel list structure:

Bit	Name	Purpose	Comments
31	LNCL_NEXT	Tells firmware that there is a next entry n the channel list	
21	DQ_LNCL_UPDALL	Check update line to update all DACs.	Reserved
20	DQ_LNCL_WRITE	Write data into the DAC, but do not update.	Reserved
7...0		Channel number	

3.3 Data Represent-ation

DNA-AO-332 has 16-bit straight binary data representation.

To convert voltage into an A/D representation, use the following formula:

```
Raw = (Volt+Offset)/(Span/0xFFFF),
```

where Volt is the desired level in volts.

3.4 Layer-specific Commands and Parameters

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

```
DqAdv3xxWrite()
```

This function works using underlying `DqCmdIoctl()`. It uses the `DQCMD_IOCTL` command with the `DQIOCTL_CVTCHNL` function.

When this function is called for the first time, the firmware terminates any ongoing operation on the device.

Then, the firmware parses the channel list and writes the passed values one by one.

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

Every write to the channel takes approximately 3.3µs. Thus, execution time for this function depends on the number of channels in the channel list.

3.5 Using the Layer in ACB Mode

This is a pseudo-code example that highlights the sequence of functions needed to use ACB on the 332 layer. A complete example with error checking can be found in the directory *SampleACB332*.

Note that we use the `#defines` for a 3xx for a DNA-AO-332 layer.

```
#include "PDNA.h"
// unit configuration word
#define CFG332          (DQ_LN_ENABLED \
                        |DQ_LN_ACTIVE \
                        |DQ_LN_GETRAW \
                        |DQ_LN_IRQEN \
                        |DQ_LN_CVCKSRC0 \
                        |DQ_LN_STREAMING \
                        |DQ_AO3xx_MODEFIFO
                        |DQ_AO3xx_BI10)
                        uint32 Config = CFG3xx;
```

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
             DqAcbInitOps(bcb,
                          &Config,
                          0,      //TrigSize,
                          NULL,   //pDQSETTRIG TrigMode,
                          &fCLClk,
                          0,      //float*  fCVClk,
                          &CLSize,
                          CL,
                          0,      //uint32* ScanBlock,
                          &acb);
             printf("Actual clock rate: %f\n", fCLClk);
             // Now set up events
             DqeSetEvent(bcb,
        DQ_eFrameDone|DQ_ePacketLost|DQ_eBufferError|DQ_eP
        acketOOB);
             // Allocate data buffer
             datta = dquser_allocatebuffer();
             // Pre-fill ACB with raw data
             dquser_prefillbuffer(data);
             DqAcbPutScansCopy(bcb, data, // buffer
                               bufsize,   // buffer size in
                                          //scans
                               bufsize,   // minimum size
                               &size,     // actual copied
                                          //size (from user
                                         // buffer into ACB)
                               &avail);

                                         // available free
                                          // space in buffer
```

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) {
            // fill buffer with more data
            dquser_prefillbuffer(data);
            DqAcbPutScansCopy(bcb, data,// buffer
                           bufsize,  // buffer size
                           MINRQ,   // minimum size
                              &size,   // actual
                           //copied size from
                           //user buffer into
                           //ACB &avail);
                           // available free space
                              //in buffer
        }
    }
```

STEP 5: Stop operation.

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

STEP 6: Clean up.

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

3.6 Using the 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]);
    }
    DqDmapInitOps(pBcb);
    DqeSetEvent(pBcb,
DQ_eDataAvailable|DQ_ePacketLost|DQ_eBufferError|D
Q_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("%04x ",
*(uint16*)ioffset[i]);
            }
        }
    }
Stop operation.
    DqeEnable(FALSE, &pBcb, 1, FALSE);
```

STEP 6: Clean up.

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

Appendices

A. Accessories

The following accessory items are available for use with the DNA-AO-332 layer.

DNA-CBL-62

This is a 3 ft., 62-way flat ribbon cable with one 62-pin male and one 62-pin D-sub connector. It is used to connect the DNA-AO-332 board to a 62-terminal panel such as the DNA-STP-62.

Index