Table Of Contents

(This page intentionally left blank.)

Appendix A: Configuration Utility

A.1 Introduction

Before DAQDRIVE can operate an adapter, a configuration file must be generated to specify the hardware configuration of the adapter. Three separate Windows based utility programs are provided on the DAQDRIVE Installation diskettes to generate these configuration files.

1. DAQCFGW.EXE is a utility to edit DAQDRIVE hardware adapter configuration files.
2. EXPBOARD.EXE is a utility to edit the data base defining available A/D expansion boards and their configuration parameters.
3. SIGCON.EXE is a utility to edit the data base defining available A/D channel signal conditioners and their configuration parameters.

IMPORTANT:
The DAQDRIVE configuration utilities must be used to create and /or edit the DAQDRIVE hardware configuration files. Under no circumstances should the user attempt to create and / or edit DAQDRIVE hardware configuration files directly.

A.2 Software Installation

The DAQDRIVE Configuration Utilities; DAQCFG, EXPBOARD, and SIGCON are installed into the DAQDRIVE\CONFIG directory by the DAQDRIVE installation program when the "Standard Installation" or the "Install Configuration Utility and Tutorial" option is selected. Consult the Software Installation section of the DAQDRIVE User's Manual for additional information.

DAQCFG does not allow the creation of new hardware configuration data files, but instead requires all files to be a modified version of an existing data file. The DAQDRIVE installation program installs the necessary sample hardware configuration data files (.DAT), and their associated report files (.RPT), into the DAQDRIVE\CONFIG directory along with the configuration utilities.

CAUTION:

Older versions of DAQDRIVE may not be compatible with files generated by the latest configuration utilities.

A.3 Generating A DAQDRIVE Configuration File

A.3.1 Starting The Configuration Utilities

To execute the DAQDRIVE configuration utility, simply double click on the configuration utility icon located in the DAQDRIVE program group.

A.3.2 Opening A Configuration Data File

DAQCFG does not allow the creation of new data files but instead requires all files to be a modified version of an existing data file (*.DAT). For this reason, one or more sample data files are provided on the DAQDRIVE installation diskettes. To view and / or edit a configuration data file:

1. Select **F**ile, **O**pen
2. Select the drive and directory in the corresponding list boxes.
3. Type the name of an existing configuration data file (.DAT) in the file name text box or select the file from the corresponding list box.
4. Choose **OK**.

A.3.3 General Configuration

The following configuration options are available from the **H**ardware Setup menu:

- General
- A / D Converter
- A / D Expansion Boards
- A / D Signal Conditioners
- D / A converter
- Timer
- Digital I / O
- Configuration Help

To select a subsystem for configuration, either select it from the **H**ardware Setup menu or click the associated toolbar icon. If a specific subsystem is not available on the adapter or if there are no user-definable options within that subsystem, the option can not be selected. The hardware specific appendix for the adapter being configured lists the available options.

The general configuration window is used to define the interface between the adapter and the host system. All adapter types require the general configuration options:

A.3.3.1 Base Address

The base I/O address of the adapter must be specified using the base address text box. If the adapter is PCMCIA, PCI, or Plug and Play compatible, the user may specify a base address of 0. Setting the base address to 0 instructs DAQDRIVE to determine the adapter's base address, interrupt, and DMA settings automatically each time the device is opened.

A.3.3.2 IRQ Level

The adapter's interrupt level (IRQ) must be selected from the corresponding drop-down list box. If the adapter does not support interrupts or if the base I/O address is set to 0, the interrupt list box is not displayed.

A.3.3.3 DMA Channel 1

The adapter's primary DMA channel must be selected from the corresponding drop-down list box. If the adapter does not support DMA or if the base I/O address is set to 0, the primary DMA list box is not displayed.

A.3.3.4 DMA Channel 2

The adapter's secondary DMA channel must be selected from the corresponding drop-down list box. If the adapter does not support two DMA channels, or if the base I/O address is set to 0, the secondary DMA list box is not displayed.

A.3.4 A/D Converter Configuration

The A/D converter window is used to define the configuration of the adapter's analog input channels. When these parameters define a specific jumper setting on the adapter, it is the user's responsibility to assure the adapter has been configured properly.

The Configuration **H**elp window provides information regarding hardware modification requirements (see section A.3.10). The number and type of user-definable options available in this window is dependent on the hardware installed and is discussed in the hardware specific appendix for the adapter being configured.

A.3.4.1 A/D Converters

Select the analog-to-digital (ADC) device on the A/D adapter to be configured from this list box. Most A/D adapters have only one ADC device (0).

A.3.4.2 Channels

Dialog box shows the number of A/D input channels available on the adapter in current configuration. A multiplexer (mux) feeds multiple A/D inputs back into the actual ADC device(s). The number of channels may be affected by the Input Mode.

A.3.4.3 Input Mode

Select the A/D input mode from the list box.

- Single Ended: A/D converter measures voltage from one input to ground. All A/D channels normally share common ground.
- Differential: A/D converter measures voltage across two inputs that are isolated from ground.

A.3.4.4 Signal Type

Select the signal type from the list box.

- Unipolar: ADC device reads only positive voltages.
- Bipolar: ADC device reads both positive and negative voltages.

A.3.4.5 Gain

This list box provides optional signal amplifier settings. Note that this option is only available on devices with hardware selectable gain settings. Devices with software programmable gains are configured at run-time.

A.3.5 A/D Converter Expansion Configuration

The A/D converter expansion window is used to define the configuration of any expansion adapters connected to the analog input channels. To assign an expansion board to an A/D channel click in the **Expansion Board Names** column and a choose from the drop down list box. The first A/D expansion board must always be connected to A/D channel 0, and additional expansion boards then connect to the next lowest channel.

Expansion adapters are defined in a data base using the **EXPBOARD** utility. The expansion board data base may be viewed from the **D**ataBase menu. However, to add or edit the expansion board data base this utility must be run independently (see section A.4).

The number and type of user-definable options available in this window is dependent on the hardware installed. The configuration of the expansion board defined in the **EXPBOARD** utility affects the options available here.

The parameters in this window refer only to the expansion board adapter and do not effect the A/D converter configuration of the main board. In most cases however, these two sets of parameters must be examined together. For example, a gain of 2 in the A/D converter configuration combined with a gain of 10 on the expansion board results in an overall gain of 20.

When these parameters define a specific jumper setting on the expansion board adapter, it is the user's responsibility to assure the adapter has been configured properly.

A.3.5.1 Channels

Dialog box shows the number of A/D input channels available on the expansion board in its current configuration. The values in this box are defined in the **EXPBOARD** utility. The number of channels may be effected by the Input Mode.

A.3.5.2 Input Mode

Select the A/D input mode from the list box.

- Single Ended: A/D converter measures voltage from one input to ground. All A/D channels normally share common ground.
- Differential: A/D converter measures voltage across two inputs that are isolated from ground.

A.3.5.3 Signal Type

Select the signal type from the list box.

- Unipolar: ADC device reads only positive voltages.
- Bipolar: ADC device reads both positive and negative voltages.

A.3.5.4 Gain

This list box provides optional signal amplifier settings. Note that this option is only available on devices with hardware selectable gain settings. Devices with software programmable gains are configurable at run-time.

A.3.6 A/D Signal Conditioners

A signal conditioner may be connected to any A/D main channel, and to any A/D expansion channel marked "Signal Conditioner Connectable" in the **EXPBOARD** utility (see A.4.2.11). Expansion boards are normally used in conjunction with signal conditioners, but are not required. To assign a signal conditioner to an A/D channel click in the **Signal Conditioner Name** column and a choose from the drop down list box.

Signal conditioners are defined in a data base using the **SIGCON** utility. The signal conditioner data base may be viewed from the **D**ataBase menu. However, to add or edit the signal conditioner data base this utility must be run independently (see section A.5).

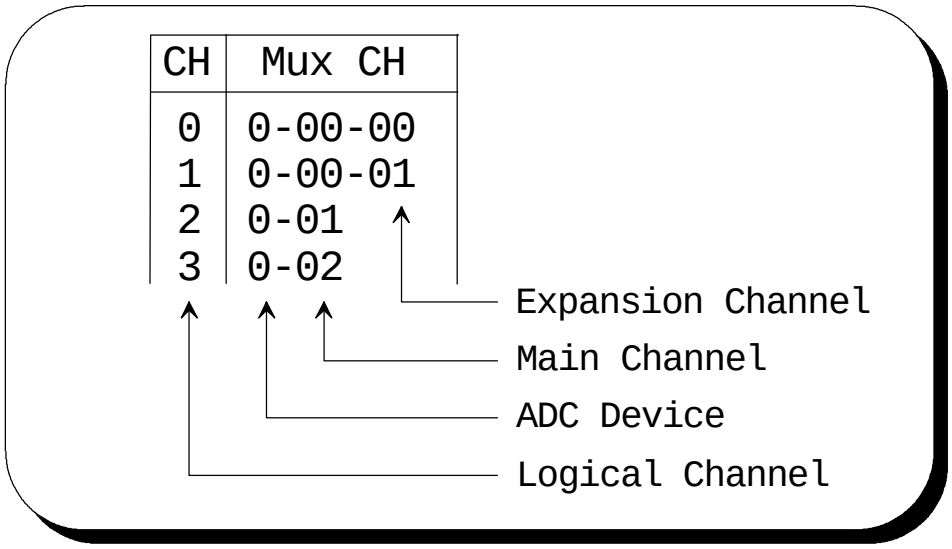

Figure 1. A/D Channel Numbering

To help understand the A/D channel numbering system the following terms are defined:

- **Logical Channel:** The CH column designates the logical number that software should use to access the analog input channel. When using expansion boards you may have up to 256 logical channels.
- **ADC Device**: The ADC device number. Most A/D adapters have only one ADC device (0).
- **Main Channel:** Analog input channel on the A/D adapter board. A multiplexer (mux) on the A/D adapter feeds multiple analog inputs back into the actual ADC device(s).
- **Expansion Channel:** Analog input channel provided by an expansion board connection to a single main analog channel. Expansion boards use digital I/O to address multiple expansion channels from a single main channel through a multiplexer.

A.3.7 D/A Converter Configuration

The D/A converter window is used to define the configuration of the adapter's analog output channels.

When these parameters define a specific jumper setting on the adapter, it is the user's responsibility to assure the adapter has been configured properly. The Configuration **H**elp window provides information regarding hardware modification requirements (see section A.3.10).

The number and type of user-definable options available in this window is dependent on the hardware installed and is discussed in the hardware specific appendix for the adapter being configured.

A.3.7.1 D/A Channels

Select the D/A channel to configure from the list. Each D/A channel typically has its own digital-to-analog converter (DAC).

A.3.7.2 Signal Type

Select the signal type from the list box.

- Unipolar: DAC device outputs only positive voltages.
- Bipolar: DAC device outputs both positive and negative voltages.

A.3.7.3 Ref. Source

Analog output from DAC is proportional to a reference voltage. Select the voltage source from the list box.

- Internal: Reference voltage generated by adapter board.
- External: Reference voltage supplied by an external source.

A.3.7.4 Ref. Voltage

The reference voltage is used as scaling multiplier for DAC output.

- For example on a 12 bit unipolar operation the analog output can be calculated from the equation:

```
A_Out = V_Ref * (Dig_Count / 4096) * Gain
```

A.3.7.5 Gain

The DAC output on some D/A adapters is connected to an output amplifier before being output to the connector. This list box provides optional output amplifier settings.

A.3.8 Timer Configuration

The timer configuration window is used to define the adapter's onboard counter / timer circuits. Examples of settings found in this section include counter size and input clock frequency.

When these parameters define a specific jumper setting on the adapter, it is the user's responsibility to assure the adapter has been configured properly. The Configuration **H**elp window provides information regarding hardware modification requirements (see section A.3.10).

The number and type of user-definable options available in this window is dependent on the hardware installed and is discussed in the hardware specific appendix for the adapter being configured.

A.3.9 Digital I/O Configuration

The digital I/O window is used to define the configuration of the adapter's digital input / output channels.

When these parameters define a specific jumper setting on the adapter, it is the user's responsibility to assure the adapter has been configured properly. The Configuration **H**elp window provides information regarding hardware modification requirements (see section A.3.10).

The number and type of user-definable options available in this window is dependent on the hardware installed and is discussed in the hardware specific appendix for the adapter being configured.

A.3.9.1 Digital Channel Configuration

Each digital I/O bit on an adapter can be individually accessed though the connector for control/monitoring of external digital devices. The digital I/O bits on each adapter must be configured into logical channels. Digital I/O channels can be set only 1 bit wide to access single I/O lines at the connector, or logical channels that access multiple I/O bits simultaneously are configurable.

Assign a logical channel number to the target digital I/O bit by clicking on the current logical channel number. A drop down channel selection box will appear with the possible channel configurations for this I/O bit (see Figure 2). The rest of the digital I/O bits will be automatically updated with correct channel numbers reflecting any changes. Repeat this step for each digital I/O bit.

A.3.9.2 Channel I/O Configuration

After all of the logical channels have been defined, they may be configured for input, output, or input/output (I/O) by clicking on the direction control button for each logical channel. All bits defined as a member of that logical channel will toggle between the available settings.

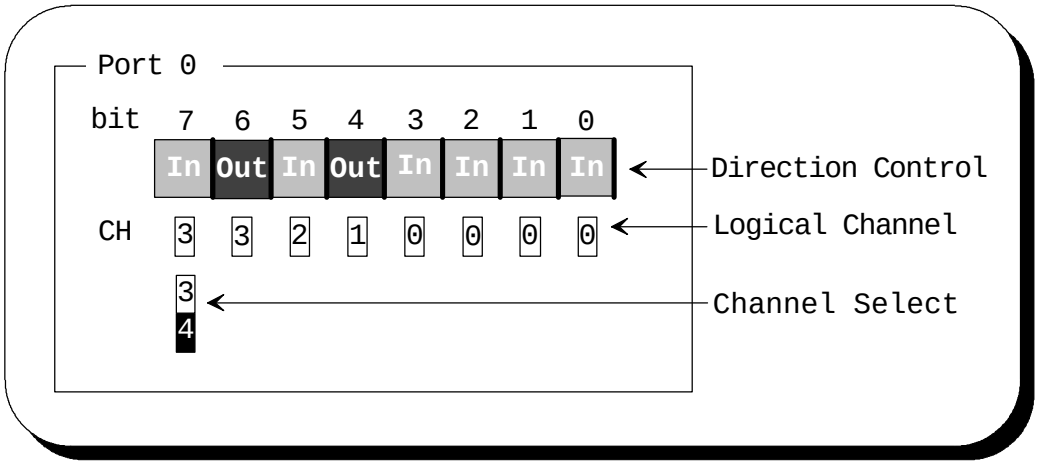

Figure 2. Digital I/O Configuration Display

A.3.10 Configuration Help

The hardware configuration of the adapter is the responsibility of the user. Some of these hardware configuration settings may be handled through software, while others may require switches or jumper blocks to be modified. The configuration help window provides the user with the jumper block or switch numbers to modify if required.

It is the responsibility of the user to determine the correct settings for the current hardware configuration. This help window is only a tool to assist the user in determining if and/or where hardware modifications are required. The amount and type of information available in this window is dependent on the hardware installed. No information is provided for configuration options handled through software.

A.3.11 Saving The New Configuration

After the adapter configuration is complete, the user may overwrite the current configuration file or a new configuration file can be generated. To overwrite the existing configuration, simply select **F**ile, **S**ave from the menu. To generate a new configuration file

1. Select **F**ile, Sa**v**e As
2. Select the drive and directory in the corresponding list boxes.
3. Type the name of the new configuration data file in the file name text box.
4. Choose **OK**.

When the user saves an adapter configuration, a corresponding report file is generated using the same file name with the extension .RPT. This report file provides an ASCII description of the hardware configuration and may be viewed using any ASCII text editor.

A.3.12 Viewing the Report File

DAQCFG also provides a utility for viewing the configuration report file (.RPT) generated when the data file was saved.

1. Select **F**ile, View **R**eport
2. Select the drive and directory in the corresponding list boxes.
3. Type the name of a report file (.RPT) in the file name text box or select a report from the corresponding list box.
4. Choose **OK**.
5. Review the adapter's configuration using the Page-Up, Page-Down, and arrow keys as well as the vertical and horizontal scroll bars.
6. When done, close the report viewer utility by selecting **C**lose.

A.4 A/D Expansion Board Database Utility

A.4.1 Starting the Expansion Board Utility

To execute the Expansion Board database configuration utility, simply double click on the expansion utility icon located in the DAQDRIVE program group.

A.4.2 Modifying the Data Base

Expansion adapters are defined in a data base using the **EXPBOARD** utility. Several adapters are predefined and may not be modified by the user. New adapters may be added or edited by the user as needed. Any modification to an expansion board configuration is automatically updated to the data base file.

The parameters for each expansion board in the data base refer only to the expansion board adapter and do not effect the A/D converter configuration of the main board. In most cases however, these two sets of parameters must be examined together. For example, a gain of 2 in the A/D converter configuration combined with a gain of 10 on the expansion board results in an overall gain of 20.

When these parameters define a specific jumper setting on the expansion board adapter, it is the user's responsibility to assure the adapter has been configured properly. Refer to the expansion board documentation for details and parameter values.

A.4.2.1 Long Device Name

Each expansion adapter must have a unique **Long Device Name** of 1 - 30 characters. The long device name used only for descriptive purposes.

A.4.2.2 Device Name

Each expansion adapter must have a unique **Device Name** of 14 characters or less.

A.4.2.3 Input Mode

Select the A/D input mode from the list box.

- Single Ended: A/D converter measures voltage from one input to ground. All A/D channels normally share common ground.
- Differential: A/D converter measures voltage across two inputs that are isolated from ground.
- DI/SE Selectable

A.4.2.4 Signal Type

Select the signal type from the list box.

- Unipolar: Expansion device reads only positive voltage.
- Bipolar: Device reads both positive and negative voltages.
- Selectable: Either of the signal types is available.

A.4.2.5 Num Gains

The **Num Gains** box refers to the number of gains available to the programmer at run-time. Therefore, if the expansion board has 4 software selectable gains this field should be set to 4. But, if the expansion board has 4 hardware (jumper or switch) selectable gains, the **Num Gains** field should be set to 1. In both cases fill in **Gains** list with all available gains.

A.4.2.6 Differential

Specify the number of differential A/D Expansion channels available on the expansion board. A typical expansion board will connect to one A/D main channel on the A/D adapter and provide up to 8 differential expansion inputs.

A.4.2.7 Single Ended

Specify the number of Single Ended A/D Expansion channels available on the expansion board. A typical expansion board will connect to one A/D main channel on the A/D adapter and provide up to 16 single ended expansion inputs.

A.4.2.8 Gains List

List all analog input amplier gains available on expansion board. When the DAQDRIVE Configuration utility is run, the A/D expansion board configuration section will include a **Gains** list box to select the expansion board gain setting if the expansion board has hardware selectable gains. Otherwise, the programmer may select any of the available expansion board gains through software at run-time.

A.4.2.9 Maximum One Channel Frequency

Maximum sampling rate for a single A/D input.

A.4.2.10 Maximum Multi-Channel Frequency

Maximum scan rate for multiple A/D inputs. Normally slower than the one channel maximum frequency since the multiplexer must switch between A/D inputs.

A.4.2.11 Channel Signal Type

Select the channel signal type from the list box. Selection determines whether the DAQDRIVE Configuration utility will allow the use of signal conditioners.

- Direct Analog Signal Connection: Expansion device supports only direct analog signal inputs.
- Signal Conditioner Connectable: Expansion device supports the use of signal conditioners.

A.5 Signal Conditioner Database Utility

A.5.1 Starting the Signal Conditioner Utility

To execute the Signal Conditioner database configuration utility, simply double click on the SIGCON utility icon located in the DAQDRIVE program group.

A.5.2 Modifying the Data Base

Signal Conditioners are defined in a data base using the **SIGCON** utility. Several entries are predefined and may not be modified by the user. New entries may be added or edited by the user as needed. Any modification to a signal conditioner configuration is automatically updated to the data base file.

The parameters for each signal conditioner in the data base refer only to the signal conditioner and do not effect the A/D converter configuration or the expansion board configuration. In most cases however, these sets of parameters must be examined together to determine the overall configuration. Refer to the signal conditioner documentation for details and parameter values.

A.5.2.1 Long Device Name

Each device must have a unique **Long Device Name** of 1 - 30 characters. The long device name used only for descriptive purposes.

A.5.2.2 Device Name

Each device must have a unique **Device Name** of 14 characters or less.

A.5.2.3 Device Type

Select the device type from the list box.

- Linear
- Nonlinear

A.5.2.4 Minimum Input

Minimum value the signal conditioner is capable of reading. Type of signal is specified by **Input Units**.

A.5.2.5 Maximum Input

Maximum value the signal conditioner is capable of reading. Type of signal is specified by **Input Units**.

A.5.2.6 Input Units

Select the measurement units for the signal type from the list box. Below is a sample of the choices.

- V (volts)
- A (amps)
- Degree C (temperature)
- Kg (kilogram)
- Hz (frequency)
- m/sec^2 (acceleration)

A.5.2.7 Minimum Output

Minimum value the signal conditioner returns to the A/D input . Type of signal is specified by **Output Units**.

A.5.2.8 Maximum Output

Maximum value the signal conditioner returns to the A/D input . Type of signal is specified by **Output Units**.

A.5.2.9 Output Units

Specifies the measurement units for the signal type returned to the A/D converter. Currently signal is always of type volts.

A.5.2.10 Bandwidth

Maximum frequency at which the signal conditioner can process data.

A.5.2.11 Maximum Scan Rate

Maximum frequency at which multiple devices may be scanned. This rate is normally slower than the bandwidth rating due to switching and settling times.

A.5.2.12 Number of Coefficients

The functional operation of a signal conditioner is defined by a polynomial equation (see Figure 3). Refer to the signal conditioner documentation for the polynomial coefficients defining the polynomial equation. The number of coefficients specified for the equation is manufacturer dependent. Specify the **Number of Coefficients** to be used in the polynomial equation in this text box.

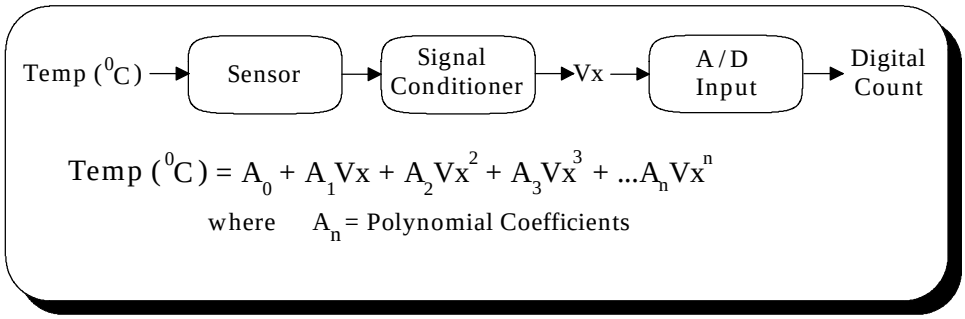

Figure 3. The Polynomial Equation

A.5.2.13 Polynomial Coefficients

Up to 12 polynomial coefficients in the polynomial equation for the signal conditioner may be specified. Fill in the **Number of Coefficients** text box to match the number of coefficient values in the **Polynomial Coefficients** list.

(This page intentionally left blank.)

Appendix B: Request Structure Tutorial

B.1 Introduction

The key to unlocking the power and versatility of DAQDRIVE is in understanding the user request structures. These data structures are responsible for configuring DAQDRIVE for the desired operation. In order to help the user understand these data structures, DAQDRIVE includes a tutorial program for Microsoft Windows, DAQTUTOR, which allows the user to define the desired operation as a series of text and option boxes and then generates the C language programming required to implement the desired operation.

B.2 Software Installation

The DAQDRIVE Request Structure Tutorial (DAQTUTOR) is installed into the DAQDRIVE\TUTOR directory by the DAQDRIVE installation program when the "Standard Installation" or the "Install Configuration Utility and Tutorial" option is selected. Consult the Software Installation section of the DAQDRIVE User's Manual for additional information.

B.3 Using The Request Structure Tutorial

B.3.1 Starting The Tutorial

To execute the DAQDRIVE request structure tutorial, simply double click on the DAQTUTOR icon located in the DAQDRIVE program group.

B.3.2 Creating An A/D Request Structure

From the menu, select **S**tructures then select **A**DC. An ADC_request window is displayed showing the default channel array, gain array, and data buffer structure settings. A second window, accessed with the Misce**l**laneous button, provides access to the additional configuration information including the trigger configuration, sampling rate, and number of scans. Additional information on A/D request structures is available in chapter 5 of the DAQDRIVE User's Manual.

B.3.2.1 Changing the channel and gain arrays

The analog input request operates on the channels specified in the channel array using the corresponding gain setting from the gain array. A valid analog input request structure requires at least one element in each of these arrays.

> **NOTE:** DAQTUTOR accepts whatever values the user places in the channel and gain arrays. It does not test the validity of the parameters nor can it determine the feasibility of the entry for any particular hardware adapter.

To add channels to the channel array:

1. Enter a channel number in the **C**hannel text box.
2. Enter a gain setting for this channel in the **G**ain text box.
3. Click **A**dd.

To remove channels from the channel array:

1. Highlight the entry in the channel array to be removed.
2. Click Re**m**ove.

B.3.2.2 Configuring the A/D data buffer structures

On the main window, DAQTUTOR displays an ADC_buffer area to configure the data buffer structures required for the analog input request. There must be at least one structure defined, therefore, the software will not allow the user to delete the default structure ADC_buf0.

To add additional data buffer structures:

1. Enter a new structure name in the **B**uffer Name text box.
2. Click **A**dd.

To remove data buffer structures from the list:

1. Highlight the entry to be removed in the **B**uffer Name window.
2. Click Re**m**ove.

For each ADC_buffer structure in the **B**uffer Name window, the user must define the following fields:

1. Status - indicates the current state of the data buffer and must be initialized to BUFFER_EMPTY for analog input requests.
2. Data Size - defines the size of each data element and must be set to the data type returned by the A/D converter.
3. Data Array - specifies the starting memory address where the A/D data will be stored.
4. L**e**ngth - defines the length of the data array in units of "number of points". The data array must be long enough to store at least one point from each channel in the channel array.
5. C**y**cles - unused for analog input requests.

NOTE: Additional details on configuring data buffer structures are provided in chapter 9 of the DAQDRIVE User's Manual.

B.3.2.3 Specifying the trigger configuration

The trigger configuration window allows the user to configure the trigger sub-system of the analog input adapter. The trigger configuration window is accessed by clicking the Miscellaneous button on the main ADC request display. The user must define the following trigger parameters:

1. Source - defines the trigger source for the request. DAQTUTOR can not confirm the availability of the requested trigger source on the target hardware adapter.
2. Mode - defines the trigger mode as one-shot (one sample/trigger) or continuous (one sample to initiate the request).
3. Slope - defines the slope of the signal required to generate the TTL or analog trigger. Slope is ignored for all other trigger sources.
4. Voltage - specifies the voltage level required for the analog trigger source. Voltage is ignored for all other trigger sources.
5. Channel - specifies the source of the trigger signal for the analog and digital trigger sources. Channel is ignored for all other trigger sources.
6. Value - specifies the value required for a digital trigger to occur. Value is ignored for all other trigger sources.

NOTE: Additional details on trigger sub-system settings are provided in chapter 10 of the DAQDRIVE User's Manual.

B.3.2.4 Miscellaneous configuration parameters

The remaining request structure parameters are accessed by clicking the Misce**l**laneous button on the main ADC request display. The user must define the following parameters:

1. **I**O mode - defines the mechanism that will be used to transfer the data from the adapter to system memory.
2. Cali**b**ration - specifies the type of calibration to be performed by the hardware during this request. DAQTUTOR can not confirm the availability of the requested calibration mode on the target hardware adapter.
3. **C**lock Source - specifies the source of the timing signal to be used for requests acquiring multiple samples.
4. **C**lock Rate - defines the frequency of the clock input when an external clock source is selected. Any value specified in this field is ignored for internal clock sources.
5. **S**ample Rate - specifies the number of samples / second (Hz) to be input from the hardware device.
6. Number of **S**cans - defines the number of times the channel(s) specified in the channel array will be processed. Each channel will be input 'Number of Scans' times.
7. Scan Event Level - defines the frequency of the scan event. A scan event is generated each time 'scan event level' scans of the channel array are completed.
8. Tim**e**out Interval - defines the amount of time DAQDRIVE will wait for an event to occur (e.g. waiting for a trigger) before abandoning the requested operation.

B.3.3 Generating A D/A Request Structure

From the menu, select **S**tructures then select **D**AC. A DAC_request window is displayed showing the default channel array and data buffer structure settings. A second window, accessed with the Misce**l**laneous button, provides access to the additional configuration information including the trigger configuration, sampling rate, and number of scans. Additional information on D/A request structures is available in chapter 6 of the DAQDRIVE User's Manual.

B.3.3.1 Changing the channel array

The analog output request operates on the channels specified in the channel array. A valid analog output request structure requires at least one element in the channel array.

> **NOTE:** DAQTUTOR accepts whatever values the user places in the channel array. It does not test the validity of the parameters nor can it determine the feasibility of the entry for any particular hardware adapter.

To add channels to the channel array:

1. Enter a channel number in the C**h**annel text box.
2. Click **A**dd.

To remove channels from the channel array:

1. Highlight the entry in the channel array to be removed.
2. Click Re**m**ove.

B.3.3.2 Configuring the D/A data buffer structures

On the main window, DAQTUTOR displays an DAC_buffer area to configure the data buffer structures required for the analog output request. There must be at least one structure defined, therefore, the software will not allow the user to delete the default structure DAC_buf0.

To add additional data buffer structures:

1. Enter a new structure name in the **B**uffer Name text box.
2. Click **A**dd.

To remove data buffer structures from the list:

1. Highlight the entry to be removed in the **B**uffer Name window.
2. Click Re**m**ove.

For each DAC_buffer structure in the **B**uffer Name window, the user must define the following fields:

1. Status - indicates the current state of the data buffer and must be initialized to BUFFER_FULL for analog output requests.
2. Data Size - defines the size of each data element and must be set to the data type required by the D/A converter.
3. Data Array - specifies the starting memory address where the D/A data is stored.
4. L**e**ngth - defines the length of the data array in units of "number of points". The data array must be long enough to contain at least one point for each channel in the channel array.
5. C**y**cles - specifies the number of times the data in this structure is processed before advancing to the next data structure. If set to 0, the current structure is processed continuously.

NOTE: Additional details on configuring data buffer structures are provided in chapter 9 of the DAQDRIVE User's Manual.

B.3.3.3 Specifying the trigger configuration

The trigger configuration window allows the user to configure the trigger sub-system of the analog output adapter. The trigger configuration window is accessed by clicking the Miscellaneous button on the main DAC request display. The user must define the following trigger parameters:

1. Source - defines the trigger source for the request. DAQTUTOR can not confirm the availability of the requested trigger source on the target hardware adapter.
2. Mode - defines the trigger mode as one-shot (one sample/trigger) or continuous (one sample to initiate the request).
3. Slope - defines the slope of the signal required to generate the TTL or analog trigger. Slope is ignored for all other trigger sources.
4. Voltage - specifies the voltage level required for the analog trigger source. Voltage is ignored for all other trigger sources.
5. Channel - specifies the source of the trigger signal for the analog and digital trigger sources. Channel is ignored for all other trigger sources.
6. Value - specifies the value required for a digital trigger to occur. Value is ignored for all other trigger sources.

NOTE: Additional details on trigger sub-system settings are provided in chapter 10 of the DAQDRIVE User's Manual.

B.3.3.4 Miscellaneous configuration parameters

The remaining request structure parameters are accessed by clicking the Misce**l**laneous button on the main DAC request display. The user must define the following parameters:

1. **I**O mode - defines the mechanism that will be used to transfer the data from system memory to the adapter.
2. Cali**b**ration - specifies the type of calibration to be performed by the hardware during this request. DAQTUTOR can not confirm the availability of the requested calibration mode on the target hardware adapter.
3. **C**lock Source - specifies the source of the timing signal to be used for requests containing multiple samples.
4. **C**lock Rate - defines the frequency of the clock input when an external clock source is selected. Any value specified in this field is ignored for internal clock sources.
5. **S**ample Rate - specifies the number of samples / second (Hz) to be output to the hardware device.
6. Number of **S**cans - defines the number of times the channel(s) specified in the channel array will be processed. Each channel will be output 'Number of Scans' times.
7. Scan Event Level - defines the frequency of the scan event. A scan event is generated each time 'scan event level' scans of the channel array are completed.
8. Tim**e**out Interval - defines the amount of time DAQDRIVE will wait for an event to occur (e.g. waiting for a trigger) before abandoning the requested operation.

B.3.4 Generating A Digital Input Request Structure

From the menu, select **S**tructures then select Digital **I**nput. A DIN_request window is displayed showing the default channel array and data buffer structure settings. A second window, accessed with the Misce**l**laneous button, provides access to the additional configuration information including the trigger configuration, sampling rate, and number of scans. Additional information on digital input request structures is available in chapter 7 of the DAQDRIVE User's Manual.

B.3.4.1 Changing the channel array

The digital input request operates on the channels specified in the channel array. A valid digital input request structure requires at least one element in the channel array.

> **NOTE:** DAQTUTOR accepts whatever values the user places in the channel array. It does not test the validity of the parameters nor can it determine the feasibility of the entry for any particular hardware adapter.

To add channels to the channel array:

1. Enter a channel number in the **C**hannel text box.
2. Click **A**dd.

To remove channels from the channel array:

1. Highlight the entry in the channel array to be removed.
2. Click Re**m**ove.

B.3.4.2 Configuring the digital input data buffer structures

On the main window, DAQTUTOR displays a DIN_buffer area to configure the data buffer structure required for the digital input request. Presently, DAQTUTOR allows only one buffer structure, DIN_buf0, for digital input requests.

For each DIN_buffer structure in the **B**uffer Name window, the user must define the following fields:

1. Status - indicates the current state of the data buffer and must be initialized to BUFFER_EMPTY for digital input requests.
2. Data Size - defines the size of each data element and must be set to the data type returned by the digital input channels.
3. Data Array - specifies the starting memory address where the digital input data will be stored.
4. L**e**ngth - defines the length of the data array in units of "number of points". The data array must be long enough to store at least one point from each channel in the channel array.
5. C**y**cles - unused for digital input requests.

NOTE: Additional details on configuring data buffer structures are provided in chapter 9 of the DAQDRIVE User's Manual.

B.3.4.3 Specifying the trigger configuration

The trigger configuration window allows the user to configure the trigger sub-system of the digital input adapter. The trigger configuration window is accessed by clicking the Miscellaneous button on the main digital input request display. The user must define the following trigger parameters:

1. Source - defines the trigger source for the request. DAQTUTOR can not confirm the availability of the requested trigger source on the target hardware adapter.
2. Mode - defines the trigger mode as one-shot (one sample/trigger) or continuous (one sample to initiate the request).
3. Slope - defines the slope of the signal required to generate the TTL or analog trigger. Slope is ignored for all other trigger sources.
4. Voltage - specifies the voltage level required for the analog trigger source. Voltage is ignored for all other trigger sources.
5. Channel - specifies the source of the trigger signal for the analog and digital trigger sources. Channel is ignored for all other trigger sources.
6. Value - specifies the value required for a digital trigger to occur. Value is ignored for all other trigger sources.

NOTE: Additional details on trigger sub-system settings are provided in chapter 10 of the DAQDRIVE User's Manual.

B.3.4.4 Miscellaneous configuration parameters

The remaining request structure parameters are accessed by clicking the Misce**l**laneous button on the main digital input request display. The user must define the following parameters:

1. **I**O mode - defines the mechanism that will be used to transfer the data from the adapter to system memory.
2. Cali**b**ration - not available with digital input request.
3. **C**lock Source - specifies the source of the timing signal to be used for requests acquiring multiple samples.
4. **C**lock Rate - defines the frequency of the clock input when an external clock source is selected. Any value specified in this field is ignored for internal clock sources.
5. **S**ample Rate - specifies the number of samples / second (Hz) to be input from the hardware device.
6. Number of **S**cans - defines the number of times the channel(s) specified in the channel array will be processed. Each channel will be input 'Number of Scans' times.
7. Scan Event Level - defines the frequency of the scan event. A scan event is generated each time 'scan event level' scans of the channel array are completed.
8. Tim**e**out Interval - defines the amount of time DAQDRIVE will wait for an event to occur (e.g. waiting for a trigger) before abandoning the requested operation.

B.3.5 Generating A Digital Output Request Structure

From the menu, select **S**tructures then select Digital **O**utput. A DOUT_request window is displayed showing the default channel array and data buffer structure settings. A second window, accessed with the Misce**l**laneous button, provides access to the additional configuration information including the trigger configuration, sampling rate, and number of scans. Additional information on digital output request structures is available in chapter 8 of the DAQDRIVE User's Manual.

B.3.5.1 Changing the channel array

The digital output request operates on the channels specified in the channel array. A valid digital output request structure requires at least one element in the channel array.

> **NOTE:** DAQTUTOR accepts whatever values the user places in the channel array. It does not test the validity of the parameters nor can it determine the feasibility of the entry for any particular hardware adapter.

To add channels to the channel array:

1. Enter a channel number in the C**h**annel text box.
2. Click **A**dd.

To remove channels from the channel array:

1. Highlight the entry in the channel array to be removed.
2. Click Re**m**ove.

B.3.5.2 Configuring the digital output data buffer structures

On the main window, DAQTUTOR displays a DOUT_buffer area to configure the data buffer structures required for the digital output request. Presently, DAQTUTOR allows only one buffer structure, DOUT_buf0, for digital output requests.

For each DOUT_buffer structure in the **B**uffer Name window, the user must define the following fields:

1. Status - indicates the current state of the data buffer and must be initialized to BUFFER_FULL for digital output requests.
2. Data Size - defines the size of each data element and must be set to the data type required by the digital output channels.
3. Data Array - specifies the starting memory address where the data to be output is stored.
4. L**e**ngth - defines the length of the data array in units of "number of points". The data array must be long enough to contain at least one point for each channel in the channel array.
5. C**y**cles - specifies the number of times the data in this structure is processed before advancing to the next data structure. If set to 0, the current structure is processed continuously.

NOTE: Additional details on configuring data buffer structures are provided in chapter 9 of the DAQDRIVE User's Manual.

B.3.5.3 Specifying the trigger configuration

The trigger configuration window allows the user to configure the trigger sub-system of the digital output adapter. The trigger configuration window is accessed by clicking the Miscellaneous button on the main digital output request display. The user must define the following trigger parameters:

1. Source - defines the trigger source for the request. DAQTUTOR can not confirm the availability of the requested trigger source on the target hardware adapter.
2. Mode - defines the trigger mode as one-shot (one sample/trigger) or continuous (one sample to initiate the request).
3. Slope - defines the slope of the signal required to generate the TTL or analog trigger. Slope is ignored for all other trigger sources.
4. Voltage - specifies the voltage level required for the analog trigger source. Voltage is ignored for all other trigger sources.
5. Channel - specifies the source of the trigger signal for the analog and digital trigger sources. Channel is ignored for all other trigger sources.
6. Value - specifies the value required for a digital trigger to occur. Value is ignored for all other trigger sources.

NOTE: Additional details on trigger sub-system settings are provided in chapter 10 of the DAQDRIVE User's Manual.

B.3.5.4 Miscellaneous configuration parameters

The remaining request structure parameters are accessed by clicking the Misce**l**laneous button on the main digital output request display. The user must define the following parameters:

1. **I**O mode - defines the mechanism that will be used to transfer the data from system memory to the adapter.
2. Cali**b**ration - not available for digital output requests.
3. **C**lock Source - specifies the source of the timing signal to be used for requests containing multiple samples.
4. **C**lock Rate - defines the frequency of the clock input when an external clock source is selected. Any value specified in this field is ignored for internal clock sources.
5. **S**ample Rate - specifies the number of samples / second (Hz) to be output to the hardware device.
6. Number of **S**cans - defines the number of times the channel(s) specified in the channel array will be processed. Each channel will be output 'Number of Scans' times.
7. Scan Event Level - defines the frequency of the scan event. A scan event is generated each time 'scan event level' scans of the channel array are completed.
8. Tim**e**out Interval - defines the amount of time DAQDRIVE will wait for an event to occur (e.g. waiting for a trigger) before abandoning the requested operation.

B.3.6 Creating Source Code

The purpose of DAQTUTOR is to allow the user to create the C language code required to initialize the DAQDRIVE request structures by simply specifying the desired configuration. Two output options are available. Option 1 copies information to and from the Windows clipboard while option two creates an ASCII file compatible with any ASCII text editor. Both of these options are discussed in the following sections.

B.3.6.1 Using the clipboard

DAQTUTOR allows the user to create the C source code to initialize the currently active request structure by simply clicking 'Co**p**y to Clipboard' at the bottom of the main display window. The user may then view this information with the Windows ClipBook Viewer application or by pasting the information into any ASCII editor (e.g. Windows Notepad). The user may also clear the clipboard at any time using the Clea**r** Clipboard button on the main display window.

B.3.6.2 Creating a source code file

DAQTUTOR allows the user to create a file containing the C source code to initialize each of the DAQDRIVE request structures. To create the source code file:

1. Select **F**ile, Sa**v**e As
2. Select the drive and directory in the corresponding list boxes.
3. Type the name of the request structure data file (.STR) in the file name text box.
4. Choose **OK**.

The resulting file contains all of the source code necessary to allocate and initialize the specified A/D, D/A, digital input, and digital output request structures. This file may be included directly into any C source file or sections may be cut-and-pasted using any ASCII text editor.

DAQTUTOR also contains a utility to allow the user to view the newly created structure file before leaving the tutorial application. To view a generated source file:

1. Select **F**ile, **V**iew File
2. Select the drive and directory in the corresponding list boxes.
3. Type the name of a structure source file (.STR) in the file name text box or select a file from the corresponding list box.
4. Choose **OK**.
5. Review the file using the Page-Up, Page-Down, and arrow keys as well as the vertical and horizontal scroll bars.
6. When done, close the viewer utility by selecting **C**lose.

(This page intentionally left blank.)

Appendix D: DAQ-16

D.1 Distribution Software

D.1.1 Creating DOS Applications Using the C Libraries

To generate an application that controls one or more DAQ-16s, the application must be linked with the appropriate DAQDRIVE library and one of the following DAQ-16 libraries:

For Microsoft Visual C/C++

- DAQ16MS.LIB - small model DAQ-16 library
- DAQ16MM.LIB - medium model DAQ-16 library
- DAQ16MC.LIB - compact model DAQ-16 library
- DAQ16ML.LIB - large model DAQ-16 library

For Borland C/C++

- DAQ16BS.LIB - small model DAQ-16 library
- DAQ16BM.LIB - medium model DAQ-16 library
- DAQ16BC.LIB - compact model DAQ-16 library
- DAQ16BL.LIB - large model DAQ-16 library

The selected libraries **MUST** match the compiler and memory model specified for the application program. These libraries are installed into the DAQDRIVE\C_LIBS directory by the DAQDRIVE installation program.

The application program must also include the file DAQ16.H installed into the DAQDRIVE\C_LIBS directory. This file defines the "open" procedure for the C library version of the DAQ-16 driver.

D.1.2 Creating DOS Applications Using The TSR Drivers

Before running a DAQ-16 application that uses the TSR drivers, the user must first load the DAQDRIVE TSR as discussed in the DAQDRIVE User's Manual. Once the DAQDRIVE TSR is installed, the user can install the DAQ-16 TSR with the command line:

```
DAQ16TSR
```

This file, DAQ16TSR.EXE, is installed into the DAQDRIVE\TSR directory by the DAQDRIVE installation program.

When the DAQ-16 TSR driver is executed, it will search for the DAQDRIVE TSR in memory and install itself on the same software interrupt. If the DAQDRIVE TSR is not loaded in memory, an error will be reported and the DAQ-16 driver will not be installed.

D.1.3 Creating Windows Applications

When a Microsoft Windows application that controls one or more DAQ-16s is executed, it must be able to dynamically link to the DAQDRIVE and DAQ-16 Dynamic Link Libraries (DLLs). Windows searches for any required DLLs in the following locations:

1. the current directory
2. the Windows directory (directory containing WIN.COM)
3. the Windows\System directory (directory containing (GDI.EXE)
4. the directory of the application program
5. all directories specified by the PATH environment variable
6. all directories mapped to network drives

The files DAQDRIVE.DLL and DAQ16.DLL are installed into the WINDOWS\SYSTEM directory by the DAQDRIVE installation program.

D.2 Configuring The DAQ-16

Before DAQDRIVE can operate the DAQ-16, a configuration data file must be generated by the DAQDRIVE configuration utility program DAQCFGW.EXE for Microsoft Windows.

D.2.1 General Configuration

The DAQ-16's base address, interrupt level and DMA channels must be defined in the general configuration window of the configuration utility. These selections must reflect the configuration of switches SW1 and SW2 and jumpers J8, J9, J10, and J11 as defined in the DAQ-16 Hardware Reference Manual.

D.2.2 A/D Converter Configuration

The DAQ-16's A/D converter must be configured for bipolar or unipolar operation and a gain value must be specified. These selections must reflect the configuration of jumpers J5 and J7 as defined in the DAQ-16 Hardware Reference Manual.

Furthermore, the data format jumper must be configured according to the input mode. If the A/D is configured for bipolar operation, the data format must be set to 2's complement using jumper J5. If the A/D is configured for unipolar operation, the data format must be set to binary using jumper J5. Jumper J6 determines input voltage range, which can be set as 10V, 5V or 2.5V. Once the input range setting is made, one should choose DAQ16_0.DAT (for 10V), DAQ16_1.DAT (for 5V), or DAQ16_2.DAT (for 2.5V) configuration data file.

D.2.3 D/A Converter Configuration

The DAQ-16's D/A converter parameters are device type (bipolar or unipolar), reference source (internal or external), and reference voltage. These selections must reflect the configuration of jumpers J3 and J4 as defined in the DAQ-16 Hardware Reference Manual.

D.2.4 Digital I/O Configuration

The DAQ-16 contains 8 bits of digital I/O. The first 4 bits are fixed output and last 4 bits are fixed input. The logical channel assignments begin with digital I/O bit 0 and continue through to digital I/O bit 7.

D.2.5 Timer Configuration

The DAQ-16 does not have any user-definable timer parameters.

D.3 Opening The DAQ-16

D.3.1 Using the DAQ-16 with the C libraries

DaqOpenDevice is the only procedure that is implemented differently depending upon the type of interface between DAQDRIVE and the application program. The C library version of DaqOpenDevice is intended for DOS applications that are written in C and linked directly to the DAQDRIVE libraries.

```
unsigned short  DaqOpenDevice(PROCEDURE,
                              unsigned short    *logical_device,
                              char    *device_type,
                              char    *config_file)
```

This version of DaqOpenDevice is implemented as a C macro and uses the token pasting operator to create a unique "open" command for the desired adapter. In order to open a DAQ-16, the application program must include DAQ16.H. In addition, the constant PROCEDURE must be replaced by DAQ16 (exactly and without quotes) and the device_type variable must be defined as "DAQ-16".

```
#include  "daqdrive.h"
#include  "daqopenc.h"
#include  "userdata.h"
#include  "daq16.h"

unsigned short main()
{
unsigned short  logical_device;
unsigned short status;

char  *device_type  = "daq-16";
char  *config_file  = "c:\\daq16\\daq-16.dat";

/*****  Open the  DAQ-16.   *****/

logical_device  = 0;
status =  DaqOpenDevice(DAQ16,   &logical_device,  device_type,  config_file);
if (status != 0)
   {
   printf("Error  opening configuration file.  Status code    %d.\n",status);
   exit(status);
   }
```

D.3.2 Using the DAQ-16 with the TSR drivers

DaqOpenDevice is the only procedure that is implemented differently depending upon the type of interface between DAQDRIVE and the application program. The TSR version of DaqOpenDevice is intended for DOS applications that interface to the memory resident (TSR) version of the DAQDRIVE drivers.

```
unsigned short  DaqOpenDevice(unsinged  short  TSR_number,
                              unsigned short    *logical_device,
                              char    *device_type,
                              char    *config_file)
```

Each hardware device supported by DAQDRIVE has been assigned a unique TSR_number value to be used with the DaqOpenDevice procedure. In order to open a DAQ-16, the TSR_number variable must be set to the value F002 hexadecimal (61, 442 decimal) and the device_type variable must be defined as "DAQ-16" for a DAQ-16 adapter.

```
#include  "daqdrive.h"
#include  "daqopent.h"
#include  "userdata.h"

unsigned short main()
{
unsigned short  logical_device;
unsigned short status;

unsigned short  TSR_number  = 0xf002;
char *device_type  = "daq-16";
char *config_file  = "c:\\daq16\\daq-16.dat";

/*****  Open the  daq-16.   *****/

logical_device  = 0;
status =  DaqOpenDevice(TSR_number,   &logical_device,  device_type,  config_file);
if (status != 0)
   {
   printf("Error  opening configuration file.  Status code    %d.\n",status);
   exit(status);
   }
```

D.3.3 Using the DAQ-16 with the Windows DLLs

DaqOpenDevice is the only procedure that is implemented differently depending upon the type of interface between DAQDRIVE and the application program. The Windows DLL version of DaqOpenDevice is intended for Windows applications that interface to the DAQDRIVE dynamic link libraries (DLLs).

```
unsigned short  DaqOpenDevice(char  *DLL_name,
                              unsigned short    *logical_device,
                              char    *device_type,
                              char    *config_file)
```

In order to open a DAQ-16, the DLL_name variable must specify the DAQ-16 dynamic link library (DAQ16.DLL) and the device_type variable must be defined as "DAQ-16".

```
#include  "daqdrive.h"
#include  "daqopenw.h"
#include  "userdata.h"

unsigned short main()
{
unsigned short  logical_device;
unsigned short status;

char  *device_type  = "DAQ-16";
char  *config_file  = "c:\\DAQ16\\DAQ-16.dat";
char  *DLL_name  = "c:\\windows\\system\\DAQ16.dll";

/*****   Open the  DAQ-16.   *****/

logical_device  = 0;
status =  DaqOpenDevice(DLL_name,   &logical_device,  device_type,  config_file);
if (status != 0)
   {
   printf("Error  opening configuration file.  Status code    %d.\n",status);
   exit(status);
   }
```

D.4 Analog Input

The DAQ-16 supports analog input requests with the following restrictions:

gain_array_ptr -	because the DAQ-16 only supports one gain setting per acquisition, all of the values specified by gain_array_ptr must be the same. In addition, the value specified for the gain must match the value stored in the DAQ-16 configuration file.
trigger_source -	only the INTERNAL_TRIGGER and TTL_TRIGGER sources are supported.
IO_mode -	DMA_FOREGROUND and DMA_BACKGROUND modes are only supported for single channel operations (i.e. when array_length = 1).
clock_source -	only the INTERNAL_CLOCK source is supported.
sample_rate -	must be in the range 2.33 mHz (2.33e-3) ≤ sample_rate ≤ 100 kHz (100e3) for single channel operation or 153 Hz ≤ sample_rate ≤ [1 / (10µsec * array_length)] for multiple channel operations (i.e. array_length > 1)
calibration -	only the NO_CALIBRATION selection is supported.

D.5 Analog Output

The DAQ-16 supports analog output requests with the following restrictions:

array_length -	only single channel operations are supported. Therefore, array_length must equal 1.
trigger_source -	only the INTERNAL_TRIGGER source is supported.
IO_mode -	only the FOREGROUND_CPU and BACKGROUND_IRQ data transfer modes are supported.
clock_source -	only the INTERNAL_CLOCK source is supported.
sample_rate -	must be greater than 153Hz. The maximum value of sample_rate is dependent upon the speed of the computer used.
calibration -	only the NO_CALIBRATION selection is supported.

D.6 Digital Input

The DAQ-16 supports digital input requests with the following restrictions:

channel_array_ptr - a channel may only appear once in the channel list.

trigger_source - only the INTERNAL_TRIGGER source is supported.

IO_mode - only the FOREGROUND_CPU data transfer mode is supported.

number_of_scans - only single point operations are supported. Therefore, number_of_scans must equal 1.

D.7 Digital Output

The DAQ-16 supports digital output requests with the following restrictions:

channel_array_ptr - a channel may only appear once in the channel list.

trigger_source - only the INTERNAL_TRIGGER is supported

IO_mode - only the FOREGROUND_CPU mode is supported.

number_of_scans - only single point operations are supported. Therefore, number_of_scans must equal 1.

(This page intentionally left blank.)

Appendix E: DAQ-801/802

E.1 Distribution Software

E.1.1 Creating DOS Applications Using The C Libraries

To generate an application that controls one or more DAQ-801/802s, the application must be linked with the appropriate DAQDRIVE library and one of the following DAQ-801/802 libraries:

For Microsoft Visual C/C++

- DAQ800MS.LIB - small model DAQ-800 library
- DAQ800MM.LIB - medium model DAQ-800 library
- DAQ800MC.LIB - compact model DAQ-800 library
- DAQ800ML.LIB - large model DAQ-800 library

For Borland C/C++

- DAQ800BS.LIB - small model DAQ-800 library
- DAQ800BM.LIB - medium model DAQ-800 library
- DAQ800BC.LIB - compact model DAQ-800 library
- DAQ800BL.LIB - large model DAQ-800 library

The selected libraries **MUST** match the compiler and memory model specified for the application program. These libraries are installed into the DAQDRIVE\C_LIBS directory by the DAQDRIVE installation program.

The application program must also include the file DAQ800.H installed into the DAQDRIVE\C_LIBS directory. This file defines the "open" procedure for the C library version of the DAQ-800 driver.

E.1.2 Creating DOS Applications Using The TSR Drivers

Before running a DAQ-801/802 application that uses the TSR drivers, the user must first load the DAQDRIVE TSR as discussed in the DAQDRIVE User's Manual. Once the DAQDRIVE TSR is installed, the user can install the DAQ-801/802 TSR with the command line:

```
DAQ-800
```

This file, DAQ-800.EXE, is installed into the DAQDRIVE\TSR directory by the DAQDRIVE installation program.

When the DAQ-801/802 TSR driver is executed, it will search for the DAQDRIVE TSR in memory and install itself on the same software interrupt. If the DAQDRIVE TSR is not loaded in memory, an error will be reported and the DAQ-801/802 driver will not be installed.

E.1.3 Creating Windows Applications

When a Microsoft Windows application that controls one or more DAQ-801/802s is executed, it must be able to dynamically link to the DAQDRIVE and DAQ-801/802 Dynamic Link Libraries (DLLs). Windows searches for any required DLLs in the following locations:

1. the current directory
2. the Windows directory (directory containing WIN.COM)
3. the Windows\System directory (directory containing (GDI.EXE)
4. the directory of the application program
5. all directories specified by the PATH environment variable
6. all directories mapped to network drives

The files DAQDRIVE.DLL and DAQ800.DLL are installed into the WINDOWS\SYSTEM directory by the DAQDRIVE installation program.

E.2 Configuring The DAQ-801/802

Before DAQDRIVE can operate the DAQ-801/802, a configuration data file must be generated by the DAQDRIVE configuration utility.

E.2.1 General Configuration

The DAQ-801/802's base address and interrupt level must be defined in the general configuration window of the configuration utility. The base address range is from 0 to 7FF0H with 10H interval. The base address value should reflect the DIP switch setting of SW1 and SW2 (refer to the DAQ-801/802 Hardware Manual).

E.2.2 A/D Converter Configuration

The only A/D converter parameter needed to be set in DAQ-801/802 is device type (Bipolar or Unipolar).

E.2.3 D/A Converter Configuration

The DAQ-801/802's D/A converter parameters are device type (bipolar or unipolar), reference source (internal or external), reference voltage, and gain (gain of 1 or 2). These parameters should reflect the jumper setting of J2 and J4 of the board (refer to the DAQ-801/802 Hardware Manual).

E.2.4 Digital I/O Configuration

The DAQ-801/802 has 32 bits of digital I/O. The first 24 bits are Port A, Port B, and Port C which are 8255A mode 0 equivalent. In the I/O port portion of the configuration window, the first 4 bits are fixed output and last 4 bits are fixed input. The 32 bits of digital I/O may be grouped into any combination of logical channels as long as the channels are in the same group type. The group type are Port A, Port B, Port C bit 0 to 3, Port C bit 4 to 7, 4-bit fixed input and 4-bit fixed output. The logical channel assignments begin with digital I/O bit 0 and continue through digital I/O bit 31.

E.2.5 Timer Configuration

The DAQ-801/802 does not have any user-definable timer parameters.

E.3 Opening The DAQ-801/802

E.3.1 Using the DAQ-801/802 with the C libraries

DaqOpenDevice is the only procedure that is implemented differently depending upon the type of interface between DAQDRIVE and the application program. The C library version of DaqOpenDevice is intended for DOS applications that are written in C and linked directly to the DAQDRIVE libraries.

```
unsigned short  DaqOpenDevice(PROCEDURE,
                              unsigned short    *logical_device,
                              char    *device_type,
                              char    *config_file)
```

This version of DaqOpenDevice is implemented as a C macro and uses the token pasting operator to create a unique "open" command for the desired adapter. In order to open a DAQ-801/802, the application program must include DAQ800.H. In addition, the constant PROCEDURE must be replaced by DAQ800 (exactly and without quotes) and the device_type variable must be defined as "DAQ-801" for a DAQ-801 adapter or "DAQ-802" for a DAQ-802 adapter.

```
#include  "daqdrive.h"
#include  "daqopenc.h"
#include  "userdata.h"
#include  "daq800.h"

unsigned short main()
{
unsigned short  logical_device;
unsigned short status;

char *device_type  = "daq-801";
char *config_file  = "c:\\daq800\\daq-801.dat";

/*****  Open the  daq-801.   *****/

logical_device  = 0;
status =  DaqOpenDevice(DAQ800,   &logical_device,  device_type,  config_file);
if (status != 0)
   {
   printf("Error  opening configuration file.  Status code    %d.\n",status);
   exit(status);
   }
```

E.3.2 Using the DAQ-801/802 with the TSR drivers

DaqOpenDevice is the only procedure that is implemented differently depending upon the type of interface between DAQDRIVE and the application program. The TSR version of DaqOpenDevice is intended for DOS applications that interface to the memory resident (TSR) version of the DAQDRIVE drivers.

```
unsigned short  DaqOpenDevice(unsigned  short  TSR_number,
                              unsigned short    *logical_device,
                              char    *device_type,
                              char    *config_file)
```

Each hardware device supported by DAQDRIVE has been assigned a unique TSR_number value to be used with the DaqOpenDevice procedure. In order to open a DAQ-801/802, the TSR_number variable must be set to the value F003 hexadecimal (61, 443 decimal) and the device_type variable must be defined as "DAQ-801" for a DAQ-801 adapter or "DAQ-802" for a DAQ-802 adapter.

```
#include  "daqdrive.h"
#include  "daqopent.h"
#include  "userdata.h"

unsigned short main()
{
unsigned short  logical_device;
unsigned short status;

unsigned short  TSR_number  = 0xf003;
char  *device_type  = "DAQ-801";
char  *config_file  = "daq-801.dat";

/*****  Open the  DAQ-801.   *****/

logical_device  = 0;
status =  DaqOpenDevice(TSR_number,   &logical_device,  device_type,  config_file);
if (status != 0)
   {
   printf("Error  opening configuration file.  Status code    %d.\n",status);
   exit(status);
   }
```

E.3.3 Using the DAQ-801/802 with the Windows DLLs

DaqOpenDevice is the only procedure that is implemented differently depending upon the type of interface between DAQDRIVE and the application program. The Windows DLL version of DaqOpenDevice is intended for Windows applications that interface to the DAQDRIVE dynamic link libraries (DLLs).

```
unsigned short  DaqOpenDevice(char  *DLL_name,
                              unsigned short    *logical_device,
                              char    *device_type,
                              char    *config_file)
```

In order to open a DAQ-801/802, the DLL_name variable must specify the DAQ-801/802 dynamic link library (DAQ800.DLL) and the device_type variable must be defined as "DAQ-801" for a DAQ-801 adapter or "DAQ-802" for a DAQ-802 adapter.

```
#include  "daqdrive.h"
#include  "daqopenw.h"
#include  "userdata.h"

unsigned short main()
{
unsigned short  logical_device;
unsigned short status;

char  *device_type  = "DAQ-802";
char  *config_file  = "c:\\DAQ800\\DAQ-802.dat";
char  *DLL_name  = "c:\\windows\\system\\DAQ800.dll";

/*****  Open the  DAQ802.   *****/

logical_device  = 0;
status =  DaqOpenDevice(DLL_name,   &logical_device,  device_type,  config_file);
if (status != 0)
   {
   printf("Error  opening configuration file.  Status code    %d.\n",status);
   exit(status);
   }
```

E.4 Analog Input

The DAQ-801/802 supports analog input requests with the following restrictions:

channel_array_ptr - In the channel array, the channel numbers must be in sequential order from start channel to stop channel. If the start channel number is greater than stop channel number, it will wrap from channel 7 to channel 0 and continue to the end channel. For example, both {2,3,4,5,6} and {6,7,0,1,2} are valid channel lists.

trigger_source - INTERNAL_TRIGGER, TTL_TRIGGER, and ANALOG_TRIGGER sources are supported.

trigger_channel - trigger_channel MUST equal the first channel in the channel list.

trigger_voltage - The trigger voltage must be within the valid analog input range of trigger_channel.

IO_mode - Only the FOREGROUND_CPU and BACKGROUND_IRQ data transfer modes are supported.

clock_source - Only the INTERNAL_CLOCK source is supported.

sample_rate - sample_rate must be in the range 5.82e-4 Hz to 40 KHz (4e4).

E.5 Analog Output

The DAQ-801/802 supports analog output requests with the following restrictions:

array_length -	only single channel operations are supported. Therefore array_length must equal 1.
trigger_source -	Only the INTERNAL_TRIGGER source is supported.
IO_mode -	Only the FOREGROUND_CPU and BACKGROUND_IRQ data transfer modes are supported.
clock_source -	Only the INTERNAL_CLOCK source is supported.
sample_rate -	The minimum value of sample_rate is 38.15Hz. The maximum value of sample_rate is depending on the speed of the computer used.
calibration -	Only the NO_CALIBRATION selection is supported.

E.6 Digital Input

The DAQ-801/802 supports digital input requests with the following restrictions:

channel_array_ptr - A channel may only appear once in the channel list.

trigger_source - Only the INTERNAL_TRIGGER source is supported.

IO_mode - Only the FOREGROUND_CPU data transfer mode is supported.

number_of_scans - Only single point operations are supported, therefore, number_of_scans must equal 1.

E.7 Digital Output

The DAQ-801/802 supports digital output requests with the following restrictions:

channel_array_ptr - A channel may only appear once in the channel list.

trigger_source - Only the INTERNAL_TRIGGER is supported

IO_mode - Only the FOREGROUND_CPU is supported.

number_of_scans - Only single point operations are supported, therefore, number_of_scans must equal 1.

(This page intentionally left blank.)

Appendix F: DAQ-1201/1202

F.1 Distribution Software

F.1.1 Creating DOS Applications Using the C Libraries

To generate an application that controls one or more DAQ-1201/1202s, the application must be linked with the appropriate DAQDRIVE library and one of the following DAQ-1201/1202 libraries:

For Microsoft Visual C/C++

- DQ1200MS.LIB - small model DAQ-1200 library
- DQ1200MM.LIB - medium model DAQ-1200 library
- DQ1200MC.LIB - compact model DAQ-1200 library
- DQ1200ML.LIB - large model DAQ-1200 library

For Borland C/C++

- DQ1200BS.LIB - small model DAQ-1200 library
- DQ1200BM.LIB - medium model DAQ-1200 library
- DQ1200BC.LIB - compact model DAQ-1200 library
- DQ1200BL.LIB - large model DAQ-1200 library

The selected libraries **MUST** match the compiler and memory model specified for the application program. These libraries are installed into the DAQDRIVE\C_LIBS directory by the DAQDRIVE installation program.

The application program must also include the file DAQ1200.H installed into the DAQDRIVE\C_LIBS directory. This file defines the "open" procedure for the C library version of the DAQ-1200 driver.

F.1.2 Creating DOS Applications Using The TSR Drivers

Before running a DAQ-1201/1202 application that uses the TSR drivers, the user must first load the DAQDRIVE TSR as discussed in the DAQDRIVE User's Manual. Once the DAQDRIVE TSR is installed, the user can install the DAQ-1200 TSR with the command line:

```
DAQ-1200
```

This file, DAQ-1200.EXE, is installed into the DAQDRIVE\TSR directory by teh DAQDRIVE installation program.

When the DAQ-1200 TSR driver is executed, it will search for the DAQDRIVE TSR in memory and install itself on the same software interrupt. If the DAQDRIVE TSR is not loaded in memory, an error will be reported and the DAQ-1200 driver will not be installed.

F.1.3 Creating Windows Applications

When a Microsoft Windows application that controls one or more DAQ-1201/1202s is executed, it must be able to dynamically link to DAQDRIVE and DAQ-1200 Dynamic Link Libraries (DLLs). Windows searches for any required DLLs in the following locations:

1. the current directory
2. the Windows directory (directory containing WIN.COM)
3. the Windows\System directory (directory containing (GDI.EXE)
4. the directory of the application program
5. all directories specified by the PATH environment variable
6. all directories mapped to network drives

The files DAQDRIVE.DLL and DAQ1200.DLL are installed into the WINDOWS\SYSTEM directory by the DAQDRIVE installation program.

F.2 Configuring The DAQ-1201/1202

Before DAQDRIVE can operate the DAQ-1201/1202, a configuration data file must be generated by the DAQDRIVE configuration utility program DAQCFGW.EXE for Microsoft Windows.

F.2.1 General Configuration

The DAQ-1201/1202's base address, interrupt level and DMA channels must be defined in the general configuration window of the configuration utility. The base address range is from 0 to 7FF0H with 10H interval. The base address value should reflect the DIP switch setting of SW1 and SW2 (refer to the DAQ-1201/1202 Hardware Manual).

F.2.2 A/D Converter Configuration

The A/D converter parameters in DAQ-1201/1202 are device type (Bipolar or Unipolar), differential or single-ended.

F.2.3 D/A Converter Configuration

The DAQ-1201/1202's D/A converter parameters are device type (bipolar or unipolar), reference source (internal or external), reference voltage, and gain (gain of 1 or 2). These parameters should reflect the jumper settings of J4 and J5 of the board (refer to the DAQ-1201/1202 Hardware Manual).

F.2.4 Digital I/O Configuration

The DAQ-1201/1202 has 32 bits of digital I/O. The first 24 bits are Port A, Port B, and Port C which are 8255A mode 0 equivalent. In the I/O port portion of the configuration window, the first 4 bits are fixed output and last 4 bits are fixed input. The 32 bits of digital I/O may be grouped into any combination of logical channels as long as the channels are in the same group type. The group type are Port A, Port B, Port C bit 0 to 3, Port C bit 4 to 7, 4-bit fixed input and 4-bit fixed output. The logical channel assignments begin with digital I/O bit 0 and continue through digital I/O bit 31.

F.2.5 Timer Configuration

The DAQ-1201/1202 does not have any user-definable timer parameters.

F.3 Opening The DAQ-1201/1202

F.3.1 Using the DAQ-1201/1202 with the C libraries

DaqOpenDevice is the only procedure that is implemented differently depending upon the type of interface between DAQDRIVE and the application program. The C library version of DaqOpenDevice is intended for DOS applications that are written in C and linked directly to the DAQDRIVE libraries.

```
unsigned short  DaqOpenDevice(PROCEDURE,
                              unsigned short    *logical_device,
                              char    *device_type,
                              char    *config_file)
```

This version of DaqOpenDevice is implemented as a C macro and uses the token pasting operator to create a unique "open" command for the desired adapter. In order to open a DAQ-1201/1202, the application program must include DAQ1200.H. In addition, the constant PROCEDURE must be replaced by the DAQ1200 (exactly and without quotes) and the device_type variable must be defined as "DAQ-1201" for a DAQ-1201 adapter or "DAQ-1202" for a DAQ-1202 adapter.

```
#include  "daqdrive.h"
#include  "daqopenc.h"
#include  "userdata.h"
#include  "daq1200.h"

unsigned short main()
{
unsigned short  logical_device;
unsigned short status;

char *device_type  = "daq-1201";
char *config_file  = "c:\\daq1200\\daq-1201.dat";

/*****  Open the  daq-1201.   *****/

logical_device  = 0;
status =  DaqOpenDevice(DAQ1200,   &logical_device,  device_type,  config_file);
if (status != 0)
   {
   printf("Error  opening configuration file.  Status code    %d.\n",status);
   exit(status);
   }
```

F.3.2 Using the DAQ-1201/1202 with the TSR drivers

DaqOpenDevice is the only procedure that is implemented differently depending upon the type of interface between DAQDRIVE and the application program. The TSR version of DaqOpenDevice is intended for DOS applications that interface to the memory resident (TSR) version of the DAQDRIVE drivers.

```
unsigned short  DaqOpenDevice(unsinged  short  TSR_number,
                              unsigned short    *logical_device,
                              char    *device_type,
                              char    *config_file)
```

Each hardware device supported by DAQDRIVE has been assigned a unique TSR_number value to be used with the DaqOpenDevice procedure. In order to open a DAQ-1201 / 1202, the TSR_number variable must be set to the value F004 hexadecimal (61, 444 decimal) and the device_type variable must be defined as "DAQ1201" for a DAQ-1201 adapter or "DAQ1202" for a DAQ-1202 adapter.

```
#include  "daqdrive.h"
#include  "daqopent.h"
#include  "userdata.h"

unsigned short main()
{
unsigned short  logical_device;
unsigned short status;

unsigned short  TSR_number  = 0xf004;
char *device_type  = "daq-1201";
char *config_file  = "c:\\daq1200\\daq-1201.dat";

/*****  Open the  daq-1201.   *****/

logical_device  = 0;
status =  DaqOpenDevice(TSR_number,   &logical_device,  device_type,  config_file);
if (status != 0)
   {
   printf("Error  opening configuration file.  Status code    %d.\n",status);
   exit(status);
   }
```

F.3.3 Using the DAQ-1201/1202 with the Windows DLLs

DaqOpenDevice is the only procedure that is implemented differently depending upon the type of interface between DAQDRIVE and the application program. The Windows DLL version of DaqOpenDevice is intended for Windows applications that interface to the DAQDRIVE dynamic link libraries (DLLs).

```
unsigned short  DaqOpenDevice(char  *DLL_name,
                              unsigned short    *logical_device,
                              char    *device_type,
                              char    *config_file)
```

In order to open a DAQ-1201/1202, the DLL_name variable must specify the DAQ-1201/1202 dynamic link library (DAQ1200.DLL) and the device_type variable must be defined as "DAQ-1201" for a DAQ-1201 adapter or "DAQ-1202" for a DAQ-1202 adapter.

```
#include  "daqdrive.h"
#include  "daqopenw.h"
#include  "userdata.h"

unsigned short main()
{
unsigned short  logical_device;
unsigned short status;

char  *device_type  = "DAQ-1202";
char  *config_file  = "c:\\DAQ1200\\DAQ-1202.dat";
char  *DLL_name  = "c:\\windows\\system\\DAQ1200.dll";

/*****   Open the  DAQ1202.   *****/

logical_device  = 0;
status =  DaqOpenDevice(DLL_name,   &logical_device,   device_type,  config_file);
if (status != 0)
   {
   printf("Error  opening configuration file.  Status code    %d.\n",status);
   exit(status);
   }
```

F.4 Analog Input

The DAQ-1201/1202 supports analog input requests with the following restrictions:

channel_array_ptr - Up to 512 channels array is supported.

trigger_source - INTERNAL_TRIGGER, TTL_TRIGGER, and ANALOG_TRIGGER sources are supported.

trigger_channel - trigger_channel MUST equal the first channel in the channel list.

trigger_voltage - The trigger voltage must be within the valid analog input range of trigger_channel.

clock_source - Only the INTERNAL_CLOCK source is supported.

sample_rate - sample_rate must be in the range 2.33e-5 Hz to 400 KHz (4e5).

calibration - Only the NO_CALIBRATION selection is supported.

F.5 Analog Output

The DAQ-1201/1202 supports analog output requests with the following restrictions:

array_length - only single channel operations are supported. Therefore array_length must equal 1.

trigger_source - Only the INTERNAL_TRIGGER source is supported.

clock_source - Only the INTERNAL_CLOCK source is supported.

sample_rate - The minimum value of sample_rate is 38.15Hz. The maximum value of sample_rate is depending on the speed of the computer used.

calibration - Only the NO_CALIBRATION selection is supported.

F.6 Digital Input

The DAQ-1201/1202 supports digital input requests with the following restrictions:

channel_array_ptr - A channel may only appear once in the channel list.

trigger_source - Only the INTERNAL_TRIGGER source is supported.

IO_mode - Only the FOREGROUND_CPU data transfer mode is supported.

number_of_scans - Only single point operations are supported, therefore, number_of_scans must equal 1.

F.7 Digital Output

The DAQ-1201/1202 supports digital output requests with the following restrictions:

channel_array_ptr - A channel may only appear once in the channel list.

trigger_source - Only the INTERNAL_TRIGGER is supported

IO_mode - Only the FOREGROUND_CPU is supported.

number_of_scans - Only single point operations are supported, therefore, number_of_scans must equal 1.

Appendix G: DAQP-12 and DAQP-16

G.1 Distribution Software

G.1.1 Creating DOS Applications Using the C Libraries

To generate an application that controls one or more DAQP-12 (or DAQP-16, or both) cards, the application must be linked with the appropriate DAQDRIVE library and one of the following DAQP libraries:

For Microsoft Visual C/C++

DAQP_CS.LIB	- small model DAQP library
DAQP_CM.LIB	- medium model DAQP library
DAQP_CC.LIB	- compact model DAQP library
DAQP_CL.LIB	- large model DAQP library

For Borland C/C++

DAQP_BS.LIB	- small model DAQP library
DAQP_BM.LIB	- medium model DAQP library
DAQP_BC.LIB	- compact model DAQP library
DAQP_BL.LIB	- large model DAQP library

The selected libraries **MUST** match the compiler and memory model specified for the application program. These libraries are installed into the DAQDRIVE\C_LIBS directory by the DAQDRIVE installation program.

The application program must also include the file DAQP.H installed into the DAQDRIVE\C_LIBS directory. This file defines the "open" procedure for the C library version of the DAQP driver.

G.1.2 Creating DOS Applications Using the TSR Driver

Before running a DAQP-12 (or DAQP-16) application that uses the TSR driver, the user must first load the DAQDRIVE TSR as discussed in the DAQDRIVE User's Manual. Once the DAQDRIVE TSR installed, the user may then install the DAQP TSR driver with the command line:

```
DAQPTSR
```

This file, DAQPTSR.EXE, is installed into the DAQDRIVE\TSR directory by the DAQDRIVE installation program.

When the DAQP TSR driver is executed, it will search for the DAQDRIVE TSR in memory and install itself on the same software interrupt. If the DAQDRIVE TSR is not loaded in memory, an error will be reported and the DAQP TSR driver will not be installed.

G.1.3 Creating Windows Applications

When a Microsoft Windows application that controls one or more DAQP-12 (or DAQP-16, or both) cards is executed, it must be able to dynamically link to the DAQDRIVE and DAQP Dynamic Link Libraries (DLLs). Windows searches for any required DLLs in the following locations

1. the current directory
2. the Windows directory (directory containing WIN.COM)
3. the Windows\System directory (directory containing (GDI.EXE)
4. the directory of the application program
5. all directories specified by the PATH environment variable
6. all directories mapped to network drives

The files DAQDRIVE.DLL and DAQPWIN.DLL are installed into the WINDOWS\SYSTEM directoyr by the DAQDRIVE installation program.

G.2 Configuring The DAQP-12 / DAQP-16

Before DAQDRIVE can operate the DAQP-12 (or DAQP-16), a configuration data file must be generated by the DAQDRIVE configuration utility program to generate the corresponding configuration data file (either DAQP-12.DAT or DAQP-16.DAT).

G.2.1 General Configuration

The DAQP-12's (or DAQP-16's) base address and interrupt level must be defined in the general configuration window of the configuration utility. If the base address is set to 0, DAQDRIVE will obtain the DAQP-12's (or DAQP-16's) base address and interrupt level from the PCMCIA Card and Socket Services software.

NOTE: To operate in auto-configuration mode, the system must have the DAQP Client Driver (the same one for both DAQP-12 and DAQP-16) and a version of Card and Socket Services software installed.

G.2.2 A/D Converter Configuration

For both DAQP-12 and DAQP-16, the A/D input channels are always bipolar. The differential or single-ended option can be selected with the configuration utility. The gains (1, 2, 4, and 8) for both cards are truly programmable.

G.2.3 Digital I/O Configuration

Both DAQP-12 and DAQP-16, have 4 bits of digital input and 4 bits for digital output. The default grouping is taking the 4 digital output bits as channel 0 and the 4 digital input bits as channel 1.

G.2.4 Timer Configuration

Neither DAQP-12 or DAQP-16 has any user-definable timer parameters.

G.3 Opening The DAQP-12 / DAQP-16

G.3.1 Using the DAQP-12 / DAQP-16 with the C libraries

DaqOpenDevice is the only procedure that is implemented differently depending upon the type of interface between DAQDRIVE and the application program. The C library version of DaqOpenDevice is intended for DOS applications that are written in C and linked directly to the DAQDRIVE libraries.

```
unsigned short  DaqOpenDevice(PROCEDURE,
                              unsigned short    *logical_device,
                              char    *device_type,
                              char    *config_file)
```

This version of DaqOpenDevice is implemented as a C macro and uses the token pasting operator to create a unique "open" command for the desired adapter.

G.3.1.1 Using the DAQP-12 with the C Libraries

In order to open a DAQP-12, the application program must include DAQP.H. In addition, the constant PROCEDURE must be replaced by DAQP (exactly and without quotes) and the device_type variable must be defined as "DAQP-12".

```
#include  "daqdrive .h"
#include  "daqopenc.h"
#include  "userdata.h"
#include  "daqp.h"

unsigned short main()
{
unsigned short  logical_device;
unsigned short status;

char  *device_type  = "DAQP-12 ";
char  *config_file  = "daqp-12.dat ";

/*****  Open the  DAQP-12.  *****/

logical_device  = 0;
status =  DaqOpenDevice( DAQP, &logical_device,  device_type,  config_file);
if (status != 0)
   {
   printf("Error  opening configuration file.  Status code    %d.\n",status);
   exit(status);
   }
```

G.3.1.2 Using the DAQP-16 with the C Libraries

In order to open a DAQP-16, the application program must include DAQP.H. In addition, the constant PROCEDURE must be replaced by DAQP(exactly and without quotes) and the device_type variable must be defined as "DAQP-16".

```
#include  "daqdrive .h"
#include  "daqopenc.h"
#include  "userdata.h"
#include  "daqp.h"

unsigned short main()
{
unsigned short  logical_device;
unsigned short status;

char  *device_type  = "DAQP-16";
char  *config_file  = "daqp-16.dat ";

/*****  Open the  DAQP-16.  *****/

logical_device  = 0;
status =  DaqOpenDevice( DAQP, &logical_device,  device_type,  config_file);
if (status != 0)
   {
   printf("Error  opening configuration file.  Status code    %d.\n",status);
   exit(status);
   }
```

G.3.2 Using the DAQP-12 / DAQP-16 with the DOS TSR Driver

DaqOpenDevice is the only procedure that is implemented differently depending upon the type of interface between DAQDRIVE and the application program. The DOS TSR version of DaqOpenDevice is intended for DOS applications that interface to the "Terminate & Stay memory-Resident" (TSR) version of the DAQDRIVE libraries.

```
unsigned short  DaqOpenDevice(unsigned  short  TSR_number,
                              unsigned short    *logical_device,
                              char    *device_type,
                              char    *config_file)
```

Each device supported by DAQDRIVE has been assigned a unique TSR_number value to be used with the DaqOpenDevice procedure. In order to open a DAQP-12 or DAQP-16 card, the TSR_number variable must be set to the value F005 hexadecimal (61,445 decimal). The device_type variable should be defined as either "DAQP-12" or "DAQP-16 " depending on the hardware in use.

G.3.2.1 Opening DAQP-12 with the DOS TSR Driver

The following example C code opens a DAQP-12 with the DOS TSR driver:

```
#include  "daqdrive .h"
#include  "daqopent.h"
#include  "userdata.h"

unsigned short main()
{
unsigned short  logical_device;
unsigned short status;

unsigned short  TSR_number  = 0xf005;
char  *device_type  = "DAQP-12 ";
char  *config_file  = "daqp-12.dat ";

/*****  Open the  DAQP-12.  *****/

logical_device  = 0;
status =  DaqOpenDevice(TSR_number,   &logical_device,  device_type,  config_file);
if (status != 0)
   {
   printf("Error  opening configuration file.  Status code    %d.\n",status);
   exit(status);
   }
```

G.3.2.2 Opening DAQP-16 with the DOS TSR Driver

The following example C code opens a DAQP-16 with the DOS TSR driver:

```
#include  "daqdrive .h"
#include  "daqopent.h"
#include  "userdata.h"

unsigned short main()
{
unsigned short  logical_device;
unsigned short status;

unsigned short  TSR_number  = 0xf005;
char  *device_type  = "DAQP-16";
char  *config_file  = "daqp-16.dat ";

/*****  Open the  DAQP-16.  *****/

logical_device  = 0;
status =  DaqOpenDevice(TSR_number,   &logical_device,  device_type,  config_file);
if (status != 0)
   {
   printf("Error  opening configuration file.  Status code    %d.\n",status);
   exit(status);
   }
```

G.3.3 Using the DAQP-12 / DAQP-16 with the Windows DLLs

DaqOpenDevice is the only procedure that is implemented differently depending upon the type of interface between DAQDRIVE and the application program. The Windows DLL version of DaqOpenDevice is intended for Windows applications that interface to the DAQDRIVE dynamic link libraries (DLLs).

```
unsigned short  DaqOpenDevice(char  *DLL_name,
                              unsigned short    *logical_device,
                              char    *device_type,
                              char    *config_file)
```

G.3.3.1 Opening the DAQP-12 with the Windows DLLs

In order to open a DAQP-12, the DLL_name variable must specify the DAQP dynamic link library (DAQPWIN.DLL) and the device_type variable must be defined as "DAQP-12".

```
#include  "daqdrive .h"
#include  "daqopenw.h"
#include  "userdata.h"

unsigned short main()
{
unsigned short  logical_device;
unsigned short status;

char  *device_type  = "DAQP-12 ";
char  *config_file  = "daqp-12.dat ";
char  *DLL_name  = "daqpwin.dll";

/*****  Open the  DAQP-12.  *****/

logical_device  = 0;
status =  DaqOpenDevice(DLL_name,   &logical_device,  device_type,  config_file);
if (status != 0)
   {
   printf("Error  opening configuration file.  Status code    %d.\n",status);
   exit(status);
   }
```

G.3.3.2 Opening the DAQP-16 with the Windows DLLs

In order to open a DAQP-16, the DLL_name variable must specify the DAQP dynamic link library (DAQPWIN.DLL) and the device_type variable must be defined as "DAQP-16".

```
#include  "daqdrive .h"
#include  "daqopenw.h"
#include  "userdata.h"

unsigned short main()
{
unsigned short  logical_device;
unsigned short status;

char  *device_type  = "DAQP-16 ";
char  *config_file  = "daqp-16.dat ";
char  *DLL_name  = "daqpwin.dll";

/*****  Open the  DAQP-16.  *****/

logical_device  = 0;
status =  DaqOpenDevice(DLL_name,   &logical_device,  device_type,  config_file);
if (status != 0)
   {
   printf("Error  opening configuration file.  Status code    %d.\n",status);
   exit(status);
   }
```

G.4 Analog Input

Both DAQP-12 and DAQP-16 support analog input requests with the following restrictions:

channel_array_ptr - requests operating on two or more analog input channels. There is no restrictions on the number of times an analog input channel may appear in the channel list.

trigger_source - only the INTERNAL_TRIGGER and TTL_TRIGGER sources are supported. If the TTL_TRIGGER is selected, the trigger signal must be applied on the DAQP-12's external trigger (shared with digital input bit 0) input.

IO_mode - only the FOREGROUND_CPU and BACKGROUND_IRQ data transfer modes are supported. DMA modes are NOT supported.

clock_source - both INTERNAL_CLOCK and EXTERNAL_CLOCK sources are supported. If the external clock is selected, the clock input has to be introduced from the external clock (shared with digital input bit 2) input. The minimum clock pulse width is 200 ns (or the maximum clock frequency is 5 MHz). There is no limit on the maximum clock width (or the minimum clock frequency).

sample_rate - sample_rate must NOT be over 100 kHz (100e3). If the internal clock is used, the minimum sampling rate is 0.06 Hz. The minimum sampling rate will be the external clock frequency divided by 16,777,215.

calibration - only the NO_CALIBRATION selection is supported.

When the data acquisition is in background mode, the DAQDRIVE low level driver will select EOS (end of scan) interrupt if the data flow is less than 1000 samples per second (sampling rate times number of channels in the scan list), otherwise it uses the FIFO threshold interrupt. The FIFO thershold will always be set as an integer multiple of the scan length, close enough to the half full level (either 256 or 1024 samples, depending on the card FIFO depth).

G.5 Analog Output

Neither DAQP-12 nor DAQP-16 has any analog output channels. All analog output requests will return with a function not supported error.

G.6 Digital Input

EitherDAQP-12 or DAQP-16 only supports single scan digital input requests with the following restrictions:

channel_array_ptr - a channel may only appear once in the channel list.

trigger_source - only the INTERNAL_TRIGGER source is supported.

IO_mode - only the FOREGROUND_CPU data transfer mode is supported.

G.7 Digital Output

Either DAQP-12 or DAQP-16 only supports single scan digital output requests with the following restrictions:

channel_array_ptr - a channel may only appear once in the channel list.

trigger_source - only the INTERNAL_TRIGGER source is supported.

IO_mode - only the FOREGROUND_CPU data transfer mode is supported.

(This page intentionally left blank.)

Appendix H: DA8P-12B

H.1 Distribution Software

H.1.1 Creating DOS Applications Using The C Libraries

To generate an application that controls one or more DA8P-12Bs, the application must be linked with the appropriate DAQDRIVE library and one of the following DA8P-12B libraries:

For Microsoft Visual C/C++

- DA8P12CS.LIB - small model DA8P-12B library
- DA8P12CM.LIB - medium model DA8P-12B library
- DA8P12CC.LIB - compact model DA8P-12B library
- DA8P12CL.LIB - large model DA8P-12B library

For Borland C/C++

- DA8P12BS.LIB - small model DA8P-12B library
- DA8P12BM.LIB - medium model DA8P-12B library
- DA8P12BC.LIB - compact model DA8P-12B library
- DA8P12BL.LIB - large model DA8P-12B library

The selected libraries **MUST** match the compiler and memory model specified for the application program. These libraries are installed into the DAQDRIVE\C_LIBS directory by the DAQDRIVE installation program.

The application program must also include the file DA8P-12.H installed into the DAQDRIVE\C_LIBS directory. This file defines the "open" procedure for the C library version of the DA8P-12B driver.

H.1.2 Creating DOS Applications Using The TSR Drivers

Before running a DA8P-12B application that uses the TSR drivers, the user must first load the DAQDRIVE TSR as discussed in the DAQDRIVE User's Manual. Once the DAQDRIVE TSR is installed, the user can install the DA8P-12B TSR with the command line:

```
DA8P-12
```

This file, DA8P-12.EXE, is installed into the DAQDRIVE\TSR directory by the DAQDRIVE installation program.

When the DA8P-12B TSR driver is executed, it will search for the DAQDRIVE TSR in memory and install itself on the same software interrupt. If the DAQDRIVE TSR is not loaded in memory, an error will be reported and the DA8P-12B driver will not be installed.

H.1.3 Creating Windows Applications

When a Microsoft Windows application that controls one or more DA8P-12Bs is executed, it must be able to dynamically link to the DAQDRIVE and DA8P-12B Dynamic Link Libraries (DLLs). Windows searches for any required DLLs in the following locations:

1. the current directory
2. the Windows directory (directory containing WIN.COM)
3. the Windows\System directory (directory containing (GDI.EXE)
4. the directory of the application program
5. all directories specified by the PATH environment variable
6. all directories mapped to network drives

The files DAQDRIVE.DLL and DA8P-12.DLL are installed into the WINDOWS\SYSTEM directory by the DAQDRIVE installation program.

H.2 Configuring The DA8P-12B

Before DAQDRIVE can operate the DA8P-12B, a configuration data file must be generated by the DAQDRIVE configuration utility.

H.2.1 General Configuration

The DA8P-12B's base address and interrupt level must be defined in the general configuration window of the configuration utility. If the base address is set to 0, DAQDRIVE will obtain the DA8P-12B's base address and interrupt level from the PCMCIA Card and Socket Services software.

NOTE: To operate in auto-configuration mode, the system must have the DA8P-12B's Client Driver and a version of Card and Socket Services software installed.

H.2.2 D/A Converter Configuration

The DA8P-12B does not have any user-definable D/A converter parameters. The DA8P-12BB is factory configured for 8 bipolar outputs. The DA8P-12BU is factory configured for 8 unipolar outputs.

H.2.3 Digital I/O Configuration

The DA8P-12B has 8 bits of digital I/O which may be grouped into any combination of logical channels. The logical channel assignments begin with digital I/O bit 0 and continue through digital I/O bit 7. After all of the logical channels have been defined, each channel may be individually configured for input, output, or input/output modes.

H.2.4 Timer Configuration

The DA8P-12B does not have any user-definable timer parameters.

H.3 Opening The DA8P-12B

H.3.1 Using the DA8P-12B with the C libraries

DaqOpenDevice is the only procedure that is implemented differently depending upon the type of interface between DAQDRIVE and the application program. The C library version of DaqOpenDevice is intended for DOS applications that are written in C and linked directly to the DAQDRIVE libraries.

```
unsigned short  DaqOpenDevice(PROCEDURE,
                              unsigned short    *logical_device,
                              char    *device_type,
                              char    *config_file)
```

This version of DaqOpenDevice is implemented as a C macro and uses the token pasting operator to create a unique "open" command for the desired adapter. In order to open a DA8P-12B, the application program must include DA8P-12.H. In addition, the constant PROCEDURE must be replaced by DA8P-12 (exactly and without quotes) and the device_type variable must be defined as "DA8P-12BB" for a bipolar adapter or "DA8P-12BU" for a unipolar adapter.

```
#include  "daqdrive .h"
#include  "daqopenc.h"
#include  "userdata.h"
#include  "da8p-12.h "

unsigned short main()
{
unsigned short  logical_device;
unsigned short status;

char  *device_type  = "DA8P-12B B";
char  *config_file  = "da8p-12b.dat ";

/*****   Open the  DA8P-12B .   *****/

logical_device  = 0;
status =  DaqOpenDevice( DA8P-12 , &logical_device,   device_type,  config_file);
if (status != 0)
   {
   printf("Error  opening configuration file.  Status code    %d.\n",status);
   exit(status);
   }
```

H.3.2 Using the DA8P-12B with the TSR drivers

DaqOpenDevice is the only procedure that is implemented differently depending upon the type of interface between DAQDRIVE and the application program. The TSR version of DaqOpenDevice is intended for DOS applications that interface to the memory resident (TSR) version of the DAQDRIVE drivers.

```
unsigned short  DaqOpenDevice(unsigned  short  TSR_number,
                              unsigned short    *logical_device,
                              char    *device_type,
                              char    *config_file)
```

Each hardware device supported by DAQDRIVE has been assigned a unique TSR_number value to be used with the DaqOpenDevice procedure. In order to open a DA8P-12B, the TSR_number variable must be set to the value F006 hexadecimal (61, 446 decimal) and the device_type variable must be defined as "DA8P-12BB" for a bipolar adapter or " DA8P-12BU" for a unipolar adapter.

```
#include  "DAQDRIVE .h"
#include  "daqopent.h"
#include  "userdata.h"

unsigned short main()
{
unsigned short  logical_device;
unsigned short status;

unsigned short  TSR_number  = 0xf006;
char  *device_type  = "DA8P-12B B";
char  *config_file  = "da8p-12b.dat ";

/*****  Open the  DA8P-12B .  *****/

logical_device  = 0;
status =  DaqOpenDevice(TSR_number,   &logical_device,  device_type,  config_file);
if (status != 0)
   {
   printf("Error  opening configuration file.  Status code    %d.\n",status);
   exit(status);
   }
```

H.3.3 Using the DA8P-12B with the Windows DLLs

DaqOpenDevice is the only procedure that is implemented differently depending upon the type of interface between DAQDRIVE and the application program. The Windows DLL version of DaqOpenDevice is intended for Windows applications that interface to the DAQDRIVE dynamic link libraries (DLLs).

```
unsigned short  DaqOpenDevice(char  *DLL_name,
                              unsigned short    *logical_device,
                              char    *device_type,
                              char    *config_file)
```

In order to open a DA8P-12B, the DLL_name variable must specify the DA8P-12Bdynamic link library (DA8P-12.DLL) and the device_type variable must be defined as "DA8P-12BB" for a bipolar adapter or " DA8P-12BU" for a unipolar adapter.

```
#include  "DAQDRIVE .h"
#include  "daqopenw.h"
#include  "userdata.h"

unsigned short main()
{
unsigned short  logical_device;
unsigned short status;

char  *device_type  = "DA8P-12B B";
char  *config_file  = "da8p-12b.dat ";
char  *DLL_name  = "c:\\windows\\system\\  da8p-12.dll ";

/*****   Open the  DA8P-12B .   *****/

logical_device  = 0;
status =  DaqOpenDevice(DLL_name,   &logical_device,   device_type,  config_file);
if (status != 0)
   {
   printf("Error  opening configuration file.  Status code    %d.\n",status);
   exit(status);
   }
```

H.4 Analog Input

The DA8P-12B does not support any analog input functions. All analog input requests will return with a function not supported error.

H.5 Analog Output

The DA8P-12B supports analog output requests with the following restrictions:

channel_array_ptr - requests operating on more than one analog output channel use the DA8P-12B's simultaneous output mode. This restricts channels to a single appearance in the channel list.

trigger_source - only the INTERNAL_TRIGGER and TTL_TRIGGER sources are supported. If the TTL_TRIGGER is selected for a single channel request, the trigger signal must be applied on the DA8P-12B's external event input. If the TTL_TRIGGER is specified for a multiple channel request, the trigger signal must be applied on the external load control input.

trigger_slope - only RISING_EDGE TTL triggers are supported.

IO_mode - only the FOREGROUND_CPU and BACKGROUND_IRQ data transfer modes are supported.

clock_source - only the INTERNAL_CLOCK source is supported.

sample_rate - sample_rate must be in the range 500 nHz (500e-9) to 100 KHz (100e3).

calibration - only the NO_CALIBRATION selection is supported.

H.6 Digital Input

The DA8P-12B supports digital input requests with the following restrictions:

channel_array_ptr - a channel may appear only once in the channel list.

number_of_scans - only one value may be input from each channel per request. Therefore, number_of_scans must equal 1.

trigger_source - only the INTERNAL_TRIGGER source is supported.

IO_mode - only the FOREGROUND_CPU mode is supported.

H.7 Digital Output

The DA8P-12B supports digital output requests with the following restrictions:

channel_array_ptr - a channel may appear only once in the channel list.

number_of_scans - only one value may be output to each channel per request. Therefore, number_of_scans must equal 1.

trigger_source - only the INTERNAL_TRIGGER source is supported.

IO_mode - only the FOREGROUND_CPU mode is supported.

Appendix I: DAQ-1101/1102

I.1 Distribution Software

I.1.1 Creating DOS Applications Using the C Libraries

To generate an application that controls one or more DAQ1101/1102s, the application must be linked with the appropriate DAQDRIVE library and one of the following DAQ-1101/1102 libraries:

For Microsoft Visual C/C++

- DQ1100MS.LIB - small model DAQ-1100 library
- DQ1100MM.LIB - medium model DAQ-1100 library
- DQ1100MC.LIB - compact model DAQ-1100 library
- DQ1100ML.LIB - large model DAQ-1100 library

For Borland C/C++

- DQ1100BS.LIB - small model DAQ-1100 library
- DQ1100BM.LIB - medium model DAQ-1100 library
- DQ1100BC.LIB - compact model DAQ-1100 library
- DQ1100BL.LIB - large model DAQ-1100 library

The selected libraries **MUST** match the compiler and memory model specified for the application program. These libraries are installed into the DAQDRIVE\C_LIBS directory by the DAQDRIVE installation program.

The application program must also include the file DAQ1100.H installed into the DAQDRIVE\C_LIBS directory. This file defines the "open" procedure for the C library version of the DAQ-1100 driver.

I.1.2 Creating DOS Applications Using The TSR Drivers

Before running a DAQ1101/1102 application that uses the TSR drivers, the user must first load the DAQDRIVE TSR as discussed in the DAQDRIVE User's Manual. Once the DAQDRIVE TSR is installed, the user can install the DAQ-1100 TSR with the command line:

```
DAQ-1100
```

This file, DAQ-1100.EXE, is located in the \TSR directory of the DAQ-1101/1102 distribution diskette.

When the DAQ-1100 TSR driver is executed, it will search for the DAQDRIVE TSR in memory and install itself on the same software interrupt. If the DAQDRIVE TSR is not loaded in memory, an error will be reported and the DAQ-1100 driver will not be installed.

I.1.3 Creating Windows Applications

When a Microsoft Windows application that controls one or more DAQ1101/1102s is executed, it must be able to dynamically link to the DAQDRIVE and DAQ-1100 Dynamic Link Libraries (DLLs). Windows searches for any required DLLs in the following locations:

1. the current directory
2. the Windows directory (directory containing WIN.COM)
3. the Windows\System directory (directory containing (GDI.EXE)
4. the directory of the application program
5. all directories specified by the PATH environment variable
6. all directories mapped to network drives

The files DAQDRIVE.DLL and DAQ1100.DLL are installed into the WINDOWS\SYSTEM directory by the DAQDRIVE installation program.

I.2 Configuring The DAQ-1101/1102

Before DAQDRIVE can operate the DAQ-1101/1102, a configuration data file must be generated by the DAQDRIVE configuration utility program DAQCFGW.EXE for Microsoft Windows.

I.2.1 General Configuration

The DAQ-1101/1102's base address, interrupt level and DMA channels must be defined in the general configuration window of the configuration utility. The base address range is from 0 to 7FF0H with 10H interval. The base address value should reflect the DIP switch setting of SW1 and SW2 (refer to the DAQ-1101/1102 Hardware Manual).

I.2.2 A/D Converter Configuration

The A/D converter parameters in DAQ-1101/1102 are device type (Bipolar or Unipolar), differential or single-ended.

I.2.3 D/A Converter Configuration

The DAQ-1101/1102's D/A converter parameters are device type (bipolar or unipolar), reference source (internal or external), reference voltage, and gain (gain of 1 or 2). These parameters should reflect the jumper settings of J4 and J5 of the board (refer to the DAQ-1101/1102 Hardware Manual).

I.2.4 Digital I/O Configuration

The DAQ-1101/1102 has 32 bits of digital I/O. The first 24 bits are Port A, Port B, and Port C which are 8255A mode 0 equivalent. In the I/O port portion of the configuration window, the first 4 bits are fixed output and last 4 bits are fixed input. The 32 bits of digital I/O may be grouped into any combination of logical channels as long as the channels are in the same group type. The group type are Port A, Port B, Port C bit 0 to 3, Port C bit 4 to 7, 4-bit fixed input and 4-bit fixed output. The logical channel assignments begin with digital I/O bit 0 and continue through digital I/O bit 31.

I.2.5 Timer Configuration

The DAQ-1101/1102 does not have any user-definable timer parameters.

I.3 Opening The DAQ-1101/1102

I.3.1 Using the DAQ-1101/1102 with the C libraries

DaqOpenDevice is the only procedure that is implemented differently depending upon the type of interface between DAQDRIVE and the application program. The C library version of DaqOpenDevice is intended for DOS applications that are written in C and linked directly to the DAQDRIVE libraries.

```
unsigned short  DaqOpenDevice(PROCEDURE,
                              unsigned short    *logical_device,
                              char    *device_type,
                              char    *config_file)
```

This version of DaqOpenDevice is implemented as a C macro and uses the token pasting operator to create a unique "open" command for the desired adapter. In order to open a DAQ-1101/1102, the application program must include DAQ1100.H. In addition, the constant PROCEDURE must be replaced by the DAQ1100 (exactly and without quotes) and the device_type variable must be defined as "DAQ-1101" for a DAQ-1101 adapter or "DAQ-1102" for a DAQ-1102 adapter.

```
#include  "DAQDRIVE .h"
#include  "daqopenc.h"
#include  "userdata.h"
#include  "daq1100.h"

unsigned short main()
{
unsigned short  logical_device;
unsigned short status;

char *device_type  = "daq-1101";
char *config_file  = "c:\\daq1100\\daq-1101.dat";

/*****  Open the  daq-1101.   *****/

logical_device  = 0;
status =  DaqOpenDevice(DAQ1100,   &logical_device,  device_type,  config_file);
if (status != 0)
   {
   printf("Error  opening configuration file.  Status code    %d.\n",status);
   exit(status);
   }
```

I.3.2 Using the DAQ-1101/1102 with the TSR drivers

DaqOpenDevice is the only procedure that is implemented differently depending upon the type of interface between DAQDRIVE and the application program. The TSR version of DaqOpenDevice is intended for DOS applications that interface to the memory resident (TSR) version of the DAQDRIVE drivers.

```
unsigned short  DaqOpenDevice(unsinged  short  TSR_number,
                              unsigned short    *logical_device,
                              char    *device_type,
                              char    *config_file)
```

Each hardware device supported by DAQDRIVE has been assigned a unique TSR_number value to be used with the DaqOpenDevice procedure. In order to open a DAQ-1101/1102, the TSR_number variable must be set to the value F007 hexadecimal (61, 447 decimal) and the device_type variable must be defined as "DAQ1101" for a DAQ-1101 adapter or "DAQ1102" for a DAQ-1102 adapter.

```
#include  "DAQDRIVE .h"
#include  "daqopent.h"
#include  "userdata.h"

unsigned short main()
{
unsigned short  logical_device;
unsigned short status;

unsigned short  TSR_number  = 0xf007;
char *device_type  = "daq-1101";
char *config_file  = "c:\\daq1100\\daq-1101.dat";

/*****  Open the  daq-1101.   *****/

logical_device  = 0;
status =  DaqOpenDevice(TSR_number,   &logical_device,  device_type,  config_file);
if (status != 0)
   {
   printf("Error  opening configuration file.  Status code    %d.\n",status);
   exit(status);
   }
```

I.3.3 Using the DAQ-1101/1102 with the Windows DLLs

DaqOpenDevice is the only procedure that is implemented differently depending upon the type of interface between DAQDRIVE and the application program. The Windows DLL version of DaqOpenDevice is intended for Windows applications that interface to the DAQDRIVE dynamic link libraries (DLLs).

```
unsigned short  DaqOpenDevice(char  *DLL_name,
                              unsigned short    *logical_device,
                              char    *device_type,
                              char    *config_file)
```

In order to open a DAQ-1101/1102, the DLL_name variable must specify the DAQ-1101/1102 dynamic link library (DAQ1100.DLL) and the device_type variable must be defined as "DAQ-1101" for a DAQ-1101 adapter or "DAQ-1102" for a DAQ-1102 adapter.

```
#include  "DAQDRIVE .h"
#include  "daqopenw.h"
#include  "userdata.h"

unsigned short main()
{
unsigned short  logical_device;
unsigned short status;

char  *device_type  = "DAQ-1102";
char  *config_file  = "c:\\DAQ1100\\DAQ-1102.dat";
char  *DLL_name  = "c:\\ WINDOWS\\SYSTEM \\DAQ1100.dll";

/*****   Open the  DAQ1102.   *****/

logical_device  = 0;
status =  DaqOpenDevice(DLL_name,   &logical_device,   device_type,  config_file);
if (status != 0)
   {
   printf("Error  opening configuration file.  Status code    %d.\n",status);
   exit(status);
   }
```

I.4 Analog Input

The DAQ-1101/1102 supports analog input requests with the following restrictions:

channel_array_ptr - Up to 256 channels array is supported. In the channel array, the channel numbers must be in sequential order from start channel to stop channel.

trigger_source - INTERNAL_TRIGGER, TTL_TRIGGER, and ANALOG_TRIGGER sources are supported.

trigger_channel - trigger_channel MUST equal the first channel in the channel list.

trigger_voltage - The trigger voltage must be within the valid analog input range of trigger_channel.

clock_source - Only the INTERNAL_CLOCK source is supported.

sample_rate - If the number of expansion board is 0, the scan speed is 2.7us from channel to channel, and the sample_rate must be in the range 2.33e-5 Hz to 333 Khz(3.33e5). Otherwise, the scan speed may be 2.7us, 10us, or 20.1us, deppending on the slowest signal condirioner in the expansion board.

calibration - Only the NO_CALIBRATION selection is supported.

I.5 Analog Output

The DAQ-1101/1102 supports analog output requests with the following restrictions:

array_length - only single channel operations are supported. Therefore array_length must equal 1.

trigger_source - Only the INTERNAL_TRIGGER source is supported.

clock_source - Only the INTERNAL_CLOCK source is supported.

sample_rate - The minimum value of sample_rate is 38.15Hz. The maximum value of sample_rate is depending on the speed of the computer used.

calibration - Only the NO_CALIBRATION selection is supported.

I.6 Digital Input

The DAQ-1101/1102 supports digital input requests with the following restrictions:

channel_array_ptr - A channel may only appear once in the channel list.

trigger_source - Only the INTERNAL_TRIGGER source is supported.

IO_mode - Only the FOREGROUND_CPU data transfer mode is supported.

number_of_scans - Only single point operations are supported, therefore, number_of_scans must equal 1.

I.7 Digital Output

The DAQ-1101/1102 supports digital output requests with the following restrictions:

channel_array_ptr - A channel may only appear once in the channel list.

trigger_source - Only the INTERNAL_TRIGGER is supported

IO_mode - Only the FOREGROUND_CPU is supported.

number_of_scans - Only single point operations are supported, therefore, number_of_scans must equal 1.

(This page intentionally left blank.)

Appendix M: IOP-241

M.1 Distribution Software

M.1.1 Creating DOS Applications Using the C Libraries

To generate an application that controls one or more IOP-241s, the application must be linked with the appropriate DAQDRIVE library and one of the following IOP-241 libraries:

For Microsoft Visual C/C++

- IOP241MS.LIB - small model IOP-241 library
- IOP241MM.LIB - medium model IOP-241 library
- IOP241MC.LIB - compact model IOP-241 library
- IOP241ML.LIB - large model IOP-241 library

For Borland C/C++

- IOP241BS.LIB - small model IOP-241 library
- IOP241BM.LIB - medium model IOP-241 library
- IOP241BC.LIB - compact model IOP-241 library
- IOP241BL.LIB - large model IOP-241 library

The selected libraries **MUST** match the compiler and memory model specified for the application program. These libraries are installed into the DAQDRIVE\C_LIBS directory by the DAQDRIVE installation program.

The application program must also include the file IOP241.H installed into the DAQDRIVE\C_LIBS directory. This file defines the "open" procedure for the C library version of the IOP-241 driver.

M.1.2 Creating DOS Applications Using The TSR Drivers

Before running an IOP-241 application that uses the TSR drivers, the user must first load the DAQDRIVE TSR as discussed in the DAQDRIVE User's Manual. Once the DAQDRIVE TSR is installed, the user can install the IOP-241 TSR with the command line:

```
IOP-241
```

This file, IOP-241.EXE, is installed into the DAQDRIVE\TSR directory by the DAQDRIVE installation program.

When the IOP-241 TSR driver is executed, it will search for the DAQDRIVE TSR in memory and install itself on the same software interrupt. If the DAQDRIVE TSR is not loaded in memory, an error will be reported and the IOP-241 driver will not be installed.

M.1.3 Creating Windows Applications

When a Microsoft Windows application that controls one or more IOP-241 s is executed, it must be able to dynamically link to the DAQDRIVE and IOP-241 Dynamic Link Libraries (DLLs). Windows searches for any required DLLs in the following locations:

1. the current directory
2. the Windows directory (directory containing WIN.COM)
3. the Windows\System directory (directory containing (GDI.EXE)
4. the directory of the application program
5. all directories specified by the PATH environment variable
6. all directories mapped to network drives

The files DAQDRIVE.DLL and IOP-241.DLL are installed into the WINDOWS\SYSTEM directory by the DAQDRIVE installation program.

M.2 Configuring The IOP-241

Before DAQDRIVE can operate the IOP-241, a configuration data file must be generated by the DAQDRIVE configuration utility program DAQCFGW.EXE for Microsoft Windows.

M.2.1 General Configuration

The IOP-241's base address must be defined in the general configuration window of the configuration utility. The base address range is from 0 to 3f8H with 8 interval. The defined card base address should be set using the IOP-241 Enabler or Client Driver (refer to theIOP-241 Hardware Manual).

M.2.2 Digital I/O Configuration

The IOP-241 has 24 bits of digital I/O. The 24 bits are grouped into three 8-bit ports. Each bit may be programmed as either input or output.. The 24 bits of digital I/O may be grouped into any combination of logical channels as long as the channels are in the same group with the same input or output type. The logical channel assignments begin with digital I/O bit 0 and continue through digital I/O bit 23.

M.3 Opening The IOP-241

M.3.1 Using the IOP-241 with the C libraries

DaqOpenDevice is the only procedure that is implemented differently depending upon the type of interface between DAQDRIVE and the application program. The C library version of DaqOpenDevice is intended for DOS applications that are written in C and linked directly to the DAQDRIVE libraries.

```
unsigned short  DaqOpenDevice(PROCEDURE,
                              unsigned short    *logical_device,
                              char    *device_type,
                              char    *config_file)
```

This version of DaqOpenDevice is implemented as a C macro and uses the token pasting operator to create a unique "open" command for the desired adapter. In order to open a IOP-241, the application program must include IOP241.H. In addition, the constant PROCEDURE must be replaced by the IOP241(exactly and without quotes) and the device_type variable must be defined as "IOP-241" for a IOP-241 adapter.

```
#include  "daqdrive .h"
#include  "daqopenc.h"
#include  "userdata.h"
#include  "iop241.h "

unsigned short main()
{
unsigned short  logical_device;
unsigned short status;

char  *device_type  = " IOP-241 ";
char  *config_file  = "c:\\ iop-241 \\iop-241.dat ";

/*****  Open the  IOP-241 .  *****/

logical_device  = 0;
status =  DaqOpenDevice( IOP241 , &logical_device,  device_type,  config_file);
if (status != 0)
   {
   printf("Error  opening configuration file.  Status code    %d.\n",status);
   exit(status);
   }
```

M.3.2 Using the IOP-241 with the TSR drivers

DaqOpenDevice is the only procedure that is implemented differently depending upon the type of interface between DAQDRIVE and the application program. The TSR version of DaqOpenDevice is intended for DOS applications that interface to the memory resident (TSR) version of the DAQDRIVE drivers.

```
unsigned short  DaqOpenDevice(unsinged  short  TSR_number,
                              unsigned short    *logical_device,
                              char    *device_type,
                              char    *config_file)
```

Each hardware device supported by DAQDRIVE has been assigned a unique TSR_number value to be used with the DaqOpenDevice procedure. In order to open a IOP-241, the TSR_number variable must be set to the value F00B hexadecimal (61, 451 decimal) and the device_type variable must be defined as "IOP-241" for a IOP-241 adapter.

```
#include  "daqdrive .h"
#include  "daqopent.h"
#include  "userdata.h"

unsigned short main()
{
unsigned short  logical_device;
unsigned short status;

unsigned short  TSR_number  = 0xf00b;
char  *device_type  = "IOP-241 ";
char  *config_file  = "c:\\ iop-241 \\iop-241.dat ";

/*****  Open the  IOP-241 .  *****/

logical_device  = 0;
status =  DaqOpenDevice(TSR_number,   &logical_device,  device_type,  config_file);
if (status != 0)
   {
   printf("Error  opening configuration file.  Status code    %d.\n",status);
   exit(status);
   }
```

M.3.3 Using the IOP-241 with Windows

DaqOpenDevice is the only procedure that is implemented differently depending upon the type of interface between DAQDRIVE and the application program. The Windows DLL version of DaqOpenDevice is intended for Windows applications that interface to the DAQDRIVE dynamic link libraries (DLLs).

```
unsigned short  DaqOpenDevice(char  *DLL_name,
                              unsigned short    *logical_device,
                              char    *device_type,
                              char    *config_file)
```

In order to open a IOP-241, the DLL_name variable must specify the IOP-241 dynamic link library (IOP-241.DLL) and the device_type variable must be defined as "IOP-241" for a IOP-241 adapter.

```
#include  "daqdrive .h"
#include  "daqopenw.h"
#include  "userdata.h"

unsigned short main()
{
unsigned short  logical_device;
unsigned short status;

char  *device_type  = "IOP-241 ";
char  *config_file  = "c:\\ iop-241 \\iop-241.dat ";
char  *DLL_name  = "c:\\ windows\system\\ iop-241.dll ";

/*****  Open the  IOP-241 .  *****/

logical_device  = 0;
status =  DaqOpenDevice(DLL_name,   &logical_device,  device_type,  config_file);
if (status != 0)
   {
   printf("Error  opening configuration file.  Status code    %d.\n",status);
   exit(status);
   }
```

M.4 Digital Input

The IOP-241 supports digital input requests with the following restrictions:

channel_array_ptr - A channel may only appear once in the channel list.

trigger_source - Only the INTERNAL_TRIGGER source is supported.

IO_mode - Only the FOREGROUND_CPU data transfer mode is supported.

number_of_scans - Only single point operations are supported, therefore, number_of_scans must equal 1.

M.5 Digital Output

The IOP-241 supports digital output requests with the following restrictions:

channel_array_ptr - A channel may only appear once in the channel list.

trigger_source - Only the INTERNAL_TRIGGER is supported

IO_mode - Only the FOREGROUND_CPU is supported.

number_of_scans - Only single point operations are supported, therefore, number_of_scans must equal 1.

(This page intentionally left blank.)

Appendix N: DAQP-208 and DAQP-308

N.1 Distribution Software

N.1.1 Creating DOS Applications Using the C Libraries

To generate an application that controls one or more DAQP-208(or DAQP-308, or both) cards, the application must be linked with the appropriate DAQDRIVE library and one of the following DAQP libraries:

For Microsoft Visual C/C++

DAQP_CS.LIB	- small model DAQP library
DAQP_CM.LIB	- medium model DAQP library
DAQP_CC.LIB	- compact model DAQP library
DAQP_CL.LIB	- large model DAQP library

For Borland C/C++

DAQP_BS.LIB	- small model DAQP library
DAQP_BM.LIB	- medium model DAQP library
DAQP_BC.LIB	- compact model DAQP library
DAQP_BL.LIB	- large model DAQP library

The selected libraries **MUST** match the compiler and memory model specified for the application program. These libraries are installed into the DAQDRIVE\C_LIBS directory by the DAQDRIVE installation program.

The application program must also include the file DAQP.H installed into the DAQDRIVE\C_LIBS directory. This file defines the "open" procedure for the C library version of the DAQP driver.

N.1.2 Creating DOS Applications Using the TSR Driver

Before running a DAQP-208(or DAQP-308) application that uses the TSR driver, the user must first load the DAQDRIVE TSR as discussed in the DAQDRIVE User's Manual. Once the DAQDRIVE TSR installed, the user may then install the DAQP TSR driver with the command line:

```
DAQPTSR
```

This file, DAQPTSR.EXE, is installed into the DAQDRIVE\TSR directory by the DAQDRIVE installation program.

When the DAQP TSR driver is executed, it will search for the DAQDRIVE TSR in memory and install itself on the same software interrupt. If the DAQDRIVE TSR is not loaded in memory, an error will be reported and the DAQP TSR driver will not be installed.

N.1.3 Creating Windows Applications

When a Microsoft Windows application that controls one or more DAQP-208(or DAQP-308, or both) cards is executed, it must be able to dynamically link to the DAQDRIVE and DAQP Dynamic Link Libraries (DLLs). Windows searches for any required DLLs in the following locations

1. the current directory
2. the Windows directory (directory containing WIN.COM)
3. the Windows\System directory (directory containing (GDI.EXE)
4. the directory of the application program
5. all directories specified by the PATH environment variable
6. all directories mapped to network drives

The files DAQDRIVE.DLL and DAQPWIN.DLL are installed into the WINDOWS\SYSTEM directory by the DAQDRIVE installation program.

N.2 Configuring The DAQP-208/ DAQP-308

Before DAQDRIVE can operate the DAQP-208(or DAQP-308), a configuration data file must be generated by the DAQDRIVE configuration utility program to generate the corresponding configuration data file (either DAQP-208.DAT or DAQP-308.DAT).

N.2.1 General Configuration

The DAQP-208's (or DAQP-308's) base address and interrupt level must be defined in the general configuration window of the configuration utility. If the base address is set to 0, DAQDRIVE will obtain the DAQP-208's (or DAQP-308's) base address and interrupt level from the PCMCIA Card and Socket Services software.

NOTE: To operate in auto-configuration mode, the system must have the DAQP Client Driver (the same one for bothDAQP-208 and DAQP-308) and a version of Card and Socket Services software installed.

N.2.2 A/D Converter Configuration

For both DAQP-208 and DAQP-308, the A/D input channels are always bipolar. The differential or single-ended option can be selected with the configuration utility. The gains (1, 2, 4, and 8) for both cards are truly programmable.

N.2.3 Digital I/O Configuration

Both DAQP-208 and DAQP-308 have 4 bits of digital input and 4 bits for digital output. The default grouping is taking the 4 digital output bits as channel 0 and the 4 digital input bits as channel 1.

N.2.4 Timer Configuration

Neither DAQP-208 nor DAQP-208 has any user-definable timer parameters.

N.2.5 D/A Converter Configuration

Neither DAQP-208 nor DAQP-208 has any user-definable D/A converter parameters.

N.3 Opening The DAQP-208/ DAQP-308

N.3.1 Using the DAQP-208/ DAQP-308 with the C libraries

DaqOpenDevice is the only procedure that is implemented differently depending upon the type of interface between DAQDRIVE and the application program. The C library version of DaqOpenDevice is intended for DOS applications that are written in C and linked directly to the DAQDRIVE libraries.

```
unsigned short  DaqOpenDevice(PROCEDURE,
                              unsigned short    *logical_device,
                              char    *device_type,
                              char    *config_file)
```

This version of DaqOpenDevice is implemented as a C macro and uses the token pasting operator to create a unique "open" command for the desired adapter.

N.3.1.1 Using the DAQP-208 with the C Libraries

In order to open a DAQP-208, the application program must include DAQP.H. In addition, the constant PROCEDURE must be replaced by DAQP (exactly and without quotes) and the device_type variable must be defined as "DAQP-208".

```
#include  "daqdrive .h"
#include  "daqopenc.h"
#include  "userdata.h"
#include  "daqp.h"

unsigned short main()
{
unsigned short  logical_device;
unsigned short status;

char  *device_type  = "DAQP-208";
char  *config_file  = "daqp-208.dat ";

/*****  Open the  DAQP-208.  *****/

logical_device  = 0;
status = DaqOpenDevice(  DAQP, &logical_device,  device_type,  config_file);
if (status != 0)
   {
   printf("Error  opening configuration file.  Status code    %d.\n",status);
   exit(status);
   }
```

N.3.1.2 Using the DAQP-308 with the C Libraries

In order to open a DAQP-308, the application program must include DAQP.H. In addition, the constant PROCEDURE must be replaced by DAQP (exactly and without quotes) and the device_type variable must be defined as "DAQP-308".

```
#include  "daqdrive .h"
#include  "daqopenc.h"
#include  "userdata.h"
#include  "daqp.h"

unsigned short main()
{
unsigned short  logical_device;
unsigned short status;

char  *device_type  = "DAQP-308";
char  *config_file  = "daqp-308.dat ";

/*****  Open the  DAQP-308.  *****/

logical_device  = 0;
status = DaqOpenDevice(  DAQP, &logical_device,  device_type,  config_file);
if (status != 0)
   {
   printf("Error  opening configuration file.  Status code    %d.\n",status);
   exit(status);
   }
```

N.3.2 Using the DAQP-208/ DAQP-308 with the DOS TSR Driver

DaqOpenDevice is the only procedure that is implemented differently depending upon the type of interface between DAQDRIVE and the application program. The DOS TSR version of DaqOpenDevice is intended for DOS applications that interface to the "Terminate & Stay memory-Resident" (TSR) version of the DAQDRIVE libraries.

```
unsigned short  DaqOpenDevice(unsigned  short  TSR_number,
                              unsigned short    *logical_device,
                              char    *device_type,
                              char    *config_file)
```

Each device supported by DAQDRIVE has been assigned a unique TSR_number value to be used with the DaqOpenDevice procedure. In order to open a DAQP-208 or DAQP-308 card, the TSR_number variable must be set to the value F005 hexadecimal (61,452 decimal). The device_type variable should be defined as "DAQP-208" forDAQP-208 and “DAQP-308” for DAQP-308.

N.3.2.1 Opening DAQP-208 with the DOS TSR Driver

The following example C code opens a DAQP-208 with the DOS TSR driver:

```
#include  "daqdrive .h"
#include  "daqopent.h"
#include  "userdata.h"

unsigned short main()
{
unsigned short  logical_device;
unsigned short status;

unsigned short  TSR_number  = 0xF005;
char  *device_type  = "DAQP-208";
char  *config_file  = "daqp-208.dat ";

/*****  Open the  DAQP-208.  *****/

logical_device  = 0;
status =  DaqOpenDevice(TSR_number,   &logical_device,  device_type,  config_file);
if (status != 0)
   {
   printf("Error  opening configuration file.  Status code    %d.\n",status);
   exit(status);
   }
```

N.3.2.2 Opening DAQP-308 with the DOS TSR Driver

The following example C code opens a DAQP-308 with the DOS TSR driver:

```
#include  "daqdrive .h"
#include  "daqopent.h"
#include  "userdata.h"

unsigned short main()
{
unsigned short  logical_device;
unsigned short status;

unsigned short TSR_number =   0xF005;
char  *device_type  = "DAQP-308";
char  *config_file  = "daqp-308.dat ";

/*****  Open the  DAQP-308.  *****/

logical_device  = 0;
status =  DaqOpenDevice(TSR_number,   &logical_device,  device_type,  config_file);
if (status != 0)
   {
   printf("Error  opening configuration file.  Status code    %d.\n",status);
   exit(status);
   }
```

N.3.3 Using the DAQP-208/ DAQP-308 with the Windows DLLs

DaqOpenDevice is the only procedure that is implemented differently depending upon the type of interface between DAQDRIVE and the application program. The Windows DLL version of DaqOpenDevice is intended for Windows applications that interface to the DAQDRIVE dynamic link libraries (DLLs).

```
unsigned short  DaqOpenDevice(char  *DLL_name,
                              unsigned short    *logical_device,
                              char    *device_type,
                              char    *config_file)
```

N.3.3.1 Opening the DAQP-208 with the Windows DLLs

In order to open a DAQP-208, the DLL_name variable must specify the DAQP dynamic link library (DAQPWIN.DLL) and the device_type variable must be defined as "DAQP-208".

```
#include  "daqdrive .h"
#include  "daqopenw.h"
#include  "userdata.h"

unsigned short main()
{
unsigned short  logical_device;
unsigned short status;

char  *device_type  = "DAQP-208";
char  *config_file  = "daqp-208.dat ";
char  *DLL_name  = "daqpwin.dll";

/*****  Open the  DAQP-208.  *****/

logical_device  = 0;
status =  DaqOpenDevice(DLL_name,   &logical_device,  device_type,  config_file);
if (status != 0)
   {
   printf("Error  opening configuration file.  Status code    %d.\n",status);
   exit(status);
   }
```

N.3.3.2 Opening the DAQP-308 with the Windows DLLs

In order to open a DAQP-308, the DLL_name variable must specify the DAQP dynamic link library (DAQPWIN.DLL) and the device_type variable must be defined as "DAQP-308".

```
#include  "daqdrive .h"
#include  "daqopenw.h"
#include  "userdata.h"

unsigned short main()
{
unsigned short  logical_device;
unsigned short status;

char  *device_type  = "DAQP-308";
char  *config_file  = "daqp-308.dat ";
char  *DLL_name  = "daqpwin.dll";

/*****  Open the  DAQP-308.  *****/

logical_device  = 0;
status =  DaqOpenDevice(DLL_name,   &logical_device,  device_type,  config_file);
if (status != 0)
   {
   printf("Error  opening configuration file.  Status code    %d.\n",status);
   exit(status);
   }
```

N.4 Analog Input

Both DAQP-208 and DAQP-308 support analog input requests with the following restrictions:

channel_array_ptr - requests operating on two or more analog input channels. There is no restrictions on the number of times an analog input channel may appear in the channel list.

trigger_source - only the INTERNAL_TRIGGER, TTL_TRIGGER and ANALOG_TRIGGER sources are supported.

trigger_channel - trigger_channel MUST be the first channel in the channel list.

trigger_voltage - The trigger voltage must be within the valid analog input range of the trigger_channel.

IO_mode - only the FOREGROUND_CPU and BACKGROUND_IRQ data transfer modes are supported. DMA modes are NOT supported

clock_source - both INTERNAL_CLOCK and EXTERNAL_CLOCK sources are supported. If the external clock is selected, the clock input has to be introduced from the external clock (shared with digital input bit 2) input. The minimum clock pulse width is 200 ns (or the maximum clock frequency is 5 MHz). There is no limit on the maximum clock width (or the minimum clock frequency).

sample_rate - sample_rate must NOT be over 100 kHz (100e3). If the internal clock is used, the minimum sampling rate is 0.06 Hz. The minimum sampling rate will be the external clock frequency divided by 16,777,215.

calibration - only the NO_CALIBRATION selection is supported.

N.5 Analog Output

There are two D/A channels in DAQP-208 and DAQP-308. Both of them support analog output requests with the following restrictions:

channel_array_ptr - If there are two channels in the list, they MUST be different from each other.

trigger_source - only the INTERNAL_TRIGGER and TTL_TRIGGER sources are supported.

trigger_slope - only the RISING_EDGE TTL trigger is supported.

IO_mode - only the FOREGROUNG_CPU and BACKGROUNG_IRQ data transfer modes are supported.

clock_source - both INTERNAL_CLOCK and EXTERNAL_CLOCK sources are supported. If the external clock is selected, the same restrictions will apply as decribeds in the previous section (N.4, A/D input, clock source)

sample_rate - sample_rate must NOT be over 100 kHz (100e3). If the internal clock is used, the minimum sampling rate is 15.3 Hz. The minimum sampling rate will be the external clock frequency divided by 65,535.

calibration - only the NO_CALIBRATION selection is supported.

N.6 Digital Input

Both DAQP-208 and DAQP-308 only support single scan digital input requests with the following restrictions:

channel_array_ptr - a channel may only appear once in the channel list.

trigger_source - only the INTERNAL_TRIGGER source is supported.

IO_mode - only the FOREGROUND_CPU data transfer mode is supported.

N.7 Digital Output

Both DAQP-208 and DAQP-308 only support single scan digital output requests with the following restrictions:

channel_array_ptr - a channel may only appear once in the channel list.

trigger_source - only the INTERNAL_TRIGGER source is supported.

IO_mode - only the FOREGROUND_CPU data transfer mode is supported.