DAQ

NI-DAQ® Function Reference Manual for PC Compatibles

Version 4.9.0

Data Acquisition Software for the PC

June 1996 Edition
Part Number 320499D-01

Internet Support

GPIB: `gpib.support@natinst.com`
DAQ: `daq.support@natinst.com`
VXI: `vxi.support@natinst.com`
LabVIEW: `lv.support@natinst.com`
LabWindows: `lw.support@natinst.com`
HiQ: `hiq.support@natinst.com`
VISA: `visa.support@natinst.com`

E-mail: `info@natinst.com`
FTP Site: `ftp.natinst.com`
Web Address: `http://www.natinst.com`

Bulletin Board Support

BBS United States: (512) 794-5422 or (800) 327-3077
BBS United Kingdom: 01635 551422
BBS France: 1 48 65 15 59

FaxBack Support

(512) 418-1111

Telephone Support (U.S.)

Tel: (512) 795-8248
Fax: (512) 794-5678

International Offices

Australia 03 9 879 9422, Austria 0662 45 79 90 0, Belgium 02 757 00 20, Canada (Ontario) 519 622 9310, Canada (Québec) 514 694 8521, Denmark 45 76 26 00, Finland 90 527 2321, France 1 48 14 24 24, Germany 089 741 31 30, Hong Kong 2645 3186, Italy 02 413091, Japan 03 5472 2970, Korea 02 596 7456, Mexico 95 800 010 0793, Netherlands 0348 433466, Norway 32 84 84 00, Singapore 2265886, Spain 91 640 0085, Sweden 08 730 49 70, Switzerland 056 200 51 51, Taiwan 02 377 1200, U.K. 01635 523545

National Instruments Corporate Headquarters

6504 Bridge Point Parkway Austin, TX 78730-5039 Tel: (512) 794-0100

Important Information

Warranty

Copyright

Trademarks

WARNING REGARDING MEDICAL AND CLINICAL USE OF NATIONAL INSTRUMENTS PRODUCTS

National Instruments products are not designed with components and testing intended to ensure a level of reliability suitable for use in treatment and diagnosis of humans. Applications of National Instruments products involving medical or clinical treatment can create a potential for accidental injury caused by product failure, or by errors on the part of the user or application designer. Any use or application of National Instruments products for or involving medical or clinical treatment must be performed by properly trained and qualified medical personnel, and all traditional medical safeguards, equipment, and procedures that are appropriate in the particular situation to prevent serious injury or death should always continue to be used when National Instruments products are being used. National Instruments products are NOT intended to be a substitute for any form of established process, procedure, or equipment used to monitor or safeguard human health and safety in medical or clinical treatment.

Table of Contents

Appendix A
Status Codes

Appendix B
Analog Input Channel and Gain Settings and Voltage Calculation

Appendix C
NI-DAQ Function Support

Appendix D
Customer Communication

Glossary

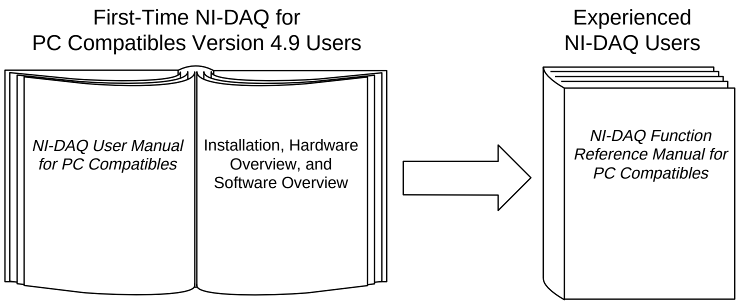
Figures

Tables

About This Manual

The *NI-DAQ Function Reference Manual for PC Compatibles* is for users of the NI-DAQ software for PC compatibles version 4.9.0. NI-DAQ software is a powerful application programming interface (API) between your data acquisition (DAQ) application and the National Instruments DAQ boards for ISA and EISA bus computers.

How to Use the NI-DAQ Manual Set


First-Time NI-DAQ for PC Compatibles Version 4.9 Users
NI-DAQ User Manual for PC Compatibles
Installation, Hardware Overview, and Software Overview
Experienced NI-DAQ Users
NI-DAQ Function Reference Manual for PC Compatibles

You should begin by reading the *NI-DAQ User Manual for PCCompatibles*. Chapter 1, *Introduction to NI-DAQ*, contains a flowchart that illustrates the sequence of steps you should take to learn about and get started with NI-DAQ.

When you are familiar with the material in the *NI-DAQ User Manual for PC Compatibles*, you can begin to use the *NI-DAQ Function Reference Manual for PC Compatibles*. The *NI-DAQ Function Reference Manual for PC Compatibles* is a reference manual that contains detailed descriptions of the NI-DAQ functions. You can also use the Windows help file `NIDAQPC.HLP`, which contains all of the function reference material.

Organization of This Manual

The *NI-DAQ Function Reference Manual for PC Compatibles* is organized as follows:

- Chapter 1, *Using the NI-DAQ Functions*, contains important information about how to apply the function descriptions in this manual to your programming language and environment.
- Chapter 2, *Function Reference*, contains a detailed explanation of each NI-DAQ function. The functions are arranged alphabetically.
- Appendix A, *Status Codes*, lists the status codes returned by NI-DAQ, including the name and description.
- Appendix B, *Analog Input Channel and Gain Settings and Voltage Calculation*, lists the valid channel and gain settings for DAQ boards, describes how NI-DAQ calculates voltage, and describes the measurement of offset and gain adjustment.
- Appendix C, *NI-DAQ Function Support*, contains tables that show which DAQ hardware each NI-DAQ function call supports.
- Appendix D, *Customer Communication*, contains forms you can use to request help from National Instruments or to comment on our products and manuals.
- The *Glossary* contains an alphabetical list and description of terms used in this manual, including abbreviations, acronyms, metric prefixes, mnemonics, and symbols.
- The *Index* contains an alphabetical list of key terms and topics in this manual, including the page where you can find each one.

Conventions Used in This Manual

The following conventions are used in this manual.

12-bit device	These MIO and AI devices do *not* have an *X* in their name, such as the AT-MIO-16 and AT-MIO-64E-3.
16-bit device	These MIO and AI devices have an *X* in their name, such as the AT-MIO-16X and AT-MIO-16XE-50.
516 device	Refers to the DAQCard-516 and PC-516.
AI device	Refers to analog input devices that have *AI* in their names, such as the NEC-AI-16E-4 (see the following *MIO and AI Device Terminology* section).

Am9513-based device	These MIO devices do *not* have an *E*- in their names. These devices are the AT-MIO-16, AT-MIO-16F-5, AT-MIO-16X, AT-MIO-16D, and AT-MIO-64F-5.
bold	Bold text denotes the names of menus, menu items, parameters, dialog boxes, dialog box buttons or options, icons, windows, Windows 95 tabs or pages, or LEDs.
bold italic	Bold italic text denotes a note, caution, or warning.
DAQCard-500/700	Refers to the DAQCard-500 and DAQCard-700.
DIO-24	Refers to the PC-DIO-24 and DAQCard-DIO-24.
DIO-32F	Refers to the AT-DIO-32F.
DIO-96	Refers to the PC-DIO-96/PnP.
DIO board	Refers to any DIO-24, DIO-32F, or DIO-96 board.
E Series device	These devices have an *E*- toward the ends of their names, such as the AT-MIO-16DE-10 and DAQPad-MIO-16XE-50.
italic	Italic text denotes emphasis, a cross reference, or an introduction to a key concept. This font also denotes text for which you supply the appropriate word or value, such as in Windows 3.*x*.
`italic monospace`	Italic text in this font denotes that you must supply the appropriate words or values in the place of these items.
Lab and 1200 Series device	Refers to the DAQCard-1200, DAQPad-1200, Lab-PC+, Lab-PC-1200, Lab-PC-1200AI, and SCXI-1200.
Lab and 1200 Series analog output device	Refers to the DAQCard-1200, DAQPad-1200, Lab-PC+, Lab-PC-1200, and SCXI-1200.
LPM device	Refers to the PC-LPM-16 and PC-LPM-16PnP.
MIO device	Refers to the multifunction I/O devices that have *MIO* in their names, such as the AT-MIO-16 and NEC-MIO-16E-4 (see the following *MIO and AI Device Terminology* section).
MIO-F-5/16X device	Refers to the AT-MIO-16F-5, AT-MIO-16X, and the AT-MIO-64F-5.
MIO-16/16D device	Refers to the AT-MIO-16 and AT-MIO-16D.
MIO-16XE-50 device	Refers to the AT-MIO-16XE-50, DAQPad-MIO-16XE-50, and NEC-MIO-16XE-50.
MIO-64	Refers to the AT-MIO-64F-5 and the AT-MIO-64E-4.
`monospace`	Text in this font denotes text or characters that you should literally enter from the keyboard, sections of code, programming examples, and

	syntax examples. This font also is used for the proper names of disk drives, paths, directories, programs, subprograms, subroutines, device names, functions, operations, variables, filenames, and extensions, and for statements and comments taken from program code.
NI-DAQ	Refers to the NI-DAQ software for PC compatibles unless otherwise noted.
Remote SCXI	Refers to an SCXI configuration where either an SCXI-2000 chassis or an SCXI-2400 remote communications module is connected to the serial port of the PC.
SCXI analog input module	Refers to the SCXI-1100, SCXI-1102, SCXI-1120, SCXI-1121, SCXI-1122, SCXI-1140, and SCXI-1141.
SCXI analog output module	Refers to the SCXI-1124.
SCXI chassis	Refers to the SCXI-1000, SCXI-1000DC, SCXI-1001, and SCXI-2000.
SCXI communication module	Refers to the SCXI-2400.
SCXI digital module	Refers to the SCXI-1160, SCXI-1161, SCXI-1162, SCXI-1162HV, SCXI-1163, and SCXI-1163R.
SCXI DAQ module	Refers to the SCXI-1200.

Abbreviations, acronyms, metric prefixes, mnemonics, symbols, and terms are listed in the *Glossary*.

MIO and AI Device Terminology

This manual uses generic terms to describe groups of devices whenever possible. The generic terms for the MIO and AI devices are based on the number of bits, the platform, the functionality, and the series name of the devices. For example, *16-bit, MIO E Series devices* refers to the AT-MIO-16XE-50, DAQPad-MIO-16XE-50, and NEC-MIO-16XE-50. Likewise, *NEC E Series devices* refers to the NEC-AI-16E-4, NEC-AI-16XE-50, NEC-MIO-16E-4, and NEC-MIO-16XE-50. The following table lists each MIO and AI device and the possible classifications for each:

Device	Bit	Type	Functionality	Series
AT-AI-16XE-10	16-bit	AT	AI	E Series
AT-MIO-16	12-bit	AT	MIO	Am9513-based
AT-MIO-16D	12-bit	AT	MIO	Am9513-based
AT-MIO-16DE-10	12-bit	AT	MIO	E Series
AT-MIO-16E-1	12-bit	AT	MIO	E Series
AT-MIO-16E-2	12-bit	AT	MIO	E Series
AT-MIO-16E-10	12-bit	AT	MIO	E Series
AT-MIO-16F-5	12-bit	AT	MIO	Am9513-based
AT-MIO-16X	16-bit	AT	MIO	Am9513-based
AT-MIO-16XE-10	16-bit	AT	MIO	E Series
AT-MIO-16XE-50	16-bit	AT	MIO	E Series
AT-MIO-64E-3	12-bit	AT	MIO	E Series
AT-MIO-64F-5	12-bit	AT	MIO	Am9513-based
DAQCard-AI-16E-4	12-bit	DAQCard	AI	E Series
DAQCard-AI-16XE-50	16-bit	DAQCard	AI	E Series
DAQPad-MIO-16XE-50	16-bit	DAQPad	MIO	E Series
NEC-AI-16E-4	12-bit	NEC	AI	E Series
NEC-AI-16XE-50	16-bit	NEC	AI	E Series
NEC-MIO-16E-4	12-bit	NEC	MIO	E Series
NEC-MIO-16XE-50	16-bit	NEC	MIO	E Series

About the National Instruments Documentation Set

The *NI-DAQ Function Reference Manual for PC Compatibles* is one piece of the documentation set for your DAQ system. You could have any of several types of manuals, depending on the hardware and software in your system. Use these manuals as follows:

- Your SCXI hardware user manuals—If you are using SCXI, read these manuals next for detailed information about signal connections and module configuration. They also explain in greater detail how the module works and contain application hints.
- Your DAQ hardware user manuals—These manuals have detailed information about the DAQ hardware that plugs into or is connected to your computer. Use these manuals for hardware installation and configuration instructions, specification information about your DAQ hardware, and application hints.
- Software manuals—Examples of software manuals you might have are the LabVIEW and LabWindows®/CVI manual sets and the NI-DAQ manuals. After you have set up your hardware system, use either the application software (LabVIEW or LabWindows/CVI) manuals or the NI-DAQ manuals to help you write your application. If you have a large and complicated system, it is worthwhile to look through the software manuals before you configure your hardware.
- Accessory installation guides or manuals—If you are using accessory products, read the terminal block and cable assembly installation guides or accessory board user manuals. They explain how to physically connect the relevant pieces of the system. Consult these guides when you are making your connections.
- SCXI chassis manuals—If you are using SCXI, read these manuals for maintenance information on the chassis and installation instructions.

Related Documentation

The following documents contain information you may find useful as you read this manual:

- *Microsoft Visual C++ User Guide to Programming*
- Omega Temperature Handbook
- NBS Monograph 125, Thermocouple Reference Tables

Customer Communication

National Instruments wants to receive your comments on our products and manuals. We are interested in the applications you develop with our products, and we want to help if you have problems with them. To make it easy for you to contact us, this manual contains comment and configuration forms for you to complete. These forms are in Appendix D, *Customer Communication*, at the end of this manual.

Chapter 1

Using the NI-DAQ Functions

This chapter contains important information about how to apply the function descriptions in this manual to your programming language and environment.

Before using this manual, you should read the NI-DAQ release notes and the *NI-DAQ User Manual for PC Compatibles*. The NI-DAQ release notes contain instructions on how to install your NI-DAQ software and how to use the online documentation set. Chapter 1, *Introduction to NI-DAQ*, in the *NI-DAQ User Manual for PC Compatibles* contains information on how to configure your National Instruments hardware and software. There is also a flowchart that illustrates the sequence of steps you should take to learn about and get started with NI-DAQ.

When you are familiar with the material in the *NI-DAQ User Manual for PC Compatibles*, you can use this manual for detailed information about each NI-DAQ function.

Status Codes

Every NI-DAQ function is of the following form:

status = `Function_Name` (parameter 1, parameter 2, … parameter *n*)

where $n \geq 0$. Each function returns a value in the status variable that indicates the success or failure of the function, as shown in Table 1-1.

Table 1-1. Status Values

Status	Result
Negative	Function did not execute because of an error
Zero	Function completed successfully
Positive	Function executed but with a potentially serious side effect

In Windows, **status** is a 2-byte integer. In Windows NT, **status** is a 4-byte integer. Appendix A, *Status Codes*, contains a list of status codes.

Variable Data Types

The NI-DAQ Application Programming Interface (API) is almost identical in Windows and Windows NT, except for some of the parameter data types. Every function description has a parameter table that lists the data types in each of the environments. LabWindows/CVI uses the same types as Windows. The following sections describe the notation used in those parameter tables and throughout the manual for variable data types.

Primary Types

Table 1-2 shows the primary type names and their ranges.

Table 1-2. Primary Type Names

Type Name	Description	Range	Type		
			C	BASIC	Pascal
U8	8-bit ASCII character	0 to 255	`char`	Not supported by BASIC. For functions that require character arrays, use string types instead. See the STR description.	`Char`
I16	16-bit signed integer	-32,768 to 32,767	`short`	Integer (for example: `deviceNum%`)	`Integer`

Table 1-2. Primary Type Names (Continued)

Type Name	Description	Range	Type		
			C	BASIC	Pascal
U16	16-bit unsigned integer	0 to 65,535	`unsigned short`	Not supported by BASIC. For functions that require unsigned integers, use the signed integer type instead. See the I16 description.	`Word`
I32	32-bit signed integer	-2,147,483,648 to 2,147,483,647	`long`	Long integer (for example: `count&`)	`Longint`
U32	32-bit unsigned integer	0 to 4,294,967,295	`unsigned long`	Not supported by BASIC. For functions that require unsigned long integers, use the signed long integer type instead. See the I32 description.	Not supported by Pascal. For functions that require unsigned long integers, use the signed long integer type instead. See the I32 description.
F32	32-bit single-precision floating point	-3.402823×10^{38} to 3.402823×10^{38}	`float`	Single-precision floating point (for example: `num!`)	`Single`
F64	64-bit double-precision floating point	$-1.797683134862315 \times 10^{308}$ to $1.797683134862315 \times 10^{308}$	`double`	Double-precision floating point (for example: `voltage#`)	`Double`
STR	BASIC or Pascal character string		Use character array terminated by the null character `\0`	Character string (for example: `filename$`)	`String`
HDL	`NI_DAQ_Mem` or DSP Memory handle	n/a	`long`	Long integer	`Longint`

Arrays

When a primary type is inside square brackets (for example, [I16]) an array of the type named is required for that parameter.

Multiple Types

Some parameters can be in multiple types. Combinations of the primary types separated by commas denote parameters with this ability, as in the following example:

[I16], HDL

The previous example describes a parameter that can accept either an array of signed integers or an `NI_DAQ_Mem` handle.

Programming Language Considerations

Apart from the data type differences, there are a few language-dependent considerations you need to be aware of when you use the NI-DAQ API. Please read the following sections that apply to your programming language.

☞ **Note:** ***Be sure to include the NI-DAQ function prototypes by including the appropriate NI-DAQ header file in your source code.***

Borland Turbo Pascal

When you pass arrays to NI-DAQ functions using Borland Turbo Pascal in Windows, you need to pass a pointer to the array. You can either declare an array and pass the array *address* to the NI-DAQ function, or you can declare a pointer, dynamically allocate memory for the pointer, and pass the pointer directly to the NI-DAQ function. For example,

```
var
   buffer : array [1..1000] of Integer;
   bufPtr : ^Integer;
status := DAQ_Start (device, chan, gain, @buffer, count,
timebase, sampInterval);
```

or

```
(* allocate memory for bufPtr first *)
status := DAQ_Start (device, chan, gain, bufPtr, count,
timebase, sampInterval);
```

Visual Basic for Windows

When you pass arrays to NI-DAQ functions using Visual Basic for Windows, you need to pass the first element of the array by reference.

For example, you would call the `DAQ_Start` function using the following syntax:

```
status% = DAQ_Start (device%, chan%, gain%, buffer%(0), count&, timebase%,
sampInterval%)
```

NI-DAQ Constants Include File

The file *`NIDAQCNS.INC`* contains definitions for constants required for some of the NI-DAQ functions. You should use the constants symbols in your programs; do not use the numerical values.

In Visual Basic for Windows, you can load the entire `NIDAQCNS.INC` file into the global module. You will then be able to use any of the constants defined in this file in any module in your program.

To do so, go to the **Project** window and choose the **Global** module, then choose **Load Text** from the **Code** menu. Select `NIDAQCNS.INC`, which is in the `NIDAQWIN\VB_EX` directory for Windows 3.1 or `NIDAQWIN95\VB_EX` directory for Windows 95. Choose **Replace** or **Merge**, depending on how you want to incorporate this file into your global module.

This procedure is identical to the procedure you would follow when loading the Visual Basic 3.0 file `CONSTANT.TXT`. Search on the word *CONSTANT* for more information from the Visual Basic on-line help. Alternatively, you can cut and paste individual lines from this file and place them in the module where you need them. However, if you do so, you should remove the word *Global* from the *CONSTANTS* definition.

For example,

```
GLOBAL CONST ND_DATA_XFER_MODE_AI&= 14000
```

would become:

```
CONST ND_DATA_XFER_MODE_AI&= 14000
```

NI-DAQ for LabWindows/CVI

Inside the LabWindows/CVI environment, the NI-DAQ functions appear in the Data Acquisition function panels under the **Libraries** menu. Each function panel represents an NI-DAQ function, which is displayed at the bottom of the panel. The function panels have help text for each function and each parameter; however, if you need additional information, you can look up the appropriate NI-DAQ function alphabetically in Chapter 2, *Function Reference*.

Table 1-3 shows how the LabWindows/CVI function panel tree is organized, and the NI-DAQ function name that corresponds to each function panel.

Table 1-3. The LabWindows/CVI Function Tree for Data Acquisition

LabWindows/CVI Function Panel	NI-DAQ Function
Data Acquisition	
Initialization/Utilities	
Initialize Board	`Init_DA_Brds`
Configure Timeout	`Timeout_Config`
Configure Master-Slave	`Master_Slave_Config`
Get Device Information	`Get_DAQ_Device_Info`
Set Device Information	`Set_DAQ_Device_Info`
Align DMA Buffer	`Align_DMA_Buffer`
Configure Trigger Window	`Trigger_Window_Config`
Get DAQ Library Version	`Get_NI_DAQ_Version`
Select E-Series Signals	`Select_Signal`
Config Analog Trigger	`Configure_HW_Analog_Trigger`
Board Config & Calibrate	
Configure A2000	`A2000_Config`
Configure DSP2200	`DSP2200_Config`
Configure MIO Boards	`MIO_Config`
Configure AMUX Boards	`AI_Mux_Config`
Configure SC-2040	`SC_2040_Config`
Calibrate A2000	`A2000_Calibrate`
Calibrate A2150	`A2150_Calibrate`
Calibrate DSP2200	`DSP2200_Calibrate`
Calibrate MIO Boards	`MIO_Calibrate`
Calibrate E-Series	`Calibrate_E_Series`
Calibrate LPM-16	`LPM16_Calibrate`
Calibrate Analog Output	`AO_Calibrate`
Calibrate 1200 Devices	`Calibrate_1200`

Table 1-3. The LabWindows/CVI Function Tree for Data Acquisition (Continued)

LabWindows/CVI Function Panel	NI-DAQ Function
Analog Input	
Single Point	
Measure Voltage	`AI_VRead`
Clear Analog Input	`AI_Clear`
Read Analog Binary	`AI_Read`
Scale Binary to Voltage	`AI_VScale`
Setup Analog Input	`AI_Setup`
Check Analog Input	`AI_Check`
Configure Analog Input	`AI_Configure`
Multiple Point	
Acquire Single Channel	`DAQ_Op`
Scan Multiple Channels	`SCAN_Op`
Scan Lab Channels	`Lab_ISCAN_Op`
Single Scan Binary	`AI_Read_Scan`
Single Scan Voltage	`AI_VRead_Scan`
Single Channel to Disk	`DAQ_to_Disk`
Multiple Chan to Disk	`SCAN_to_Disk`
Scan Lab Chan to Disk	`Lab_ISCAN_to_Disk`
Low-Level Functions	
Convert DAQ Rate	`DAQ_Rate`
Start DAQ	`DAQ_Start`
Setup Scan	`SCAN_Setup`
Setup Sequence of Scans	`SCAN_Sequence_Setup`
Retrieve Scan Sequence	`SCAN_Sequence_Retrieve`
Start Scan	`SCAN_Start`
Check DAQ or Scan	`DAQ_Check`
Monitor DAQ or Scan	`DAQ_Monitor`
Start Lab Scan	`Lab_ISCAN_Start`
Check Lab Scan	`Lab_ISCAN_Check`

Table 1-3. The LabWindows/CVI Function Tree for Data Acquisition (Continued)

LabWindows/CVI Function Panel	NI-DAQ Function
Clear DAQ or Scan	`DAQ_Clear`
Scale DAQ or Scan	`DAQ_VScale`
Reorder Scan Data	`SCAN_Demux`
Reorder Scan Seq Data	`SCAN_Sequence_Demux`
Configure DAQ	`DAQ_Config`
Config DAQ Pretrigger	`DAQ_StopTrigger_Config`
Config Double Buffering	`DAQ_DB_Config`
Is Half Buffer Ready?	`DAQ_DB_HalfReady`
Half Buffer to Array	`DAQ_DB_Transfer`
Half Buffer to String	`DAQ_DB_StrTransfer`
Simultaneous Sample/Hold	
Single-Scan (MAI)	
Select Coupling	`MAI_Coupling`
Setup Multi-Channel AI	`MAI_Setup`
Arm Multi-Channel AI	`MAI_Arm`
Read Multi-Channel AI	`MAI_Read`
Scale Multi-Channel AI	`MAI_Scale`
Clear Multi-Channel AI	`MAI_Clear`
Multiple-Scan (MDAQ)	
Select Trigger	`MDAQ_Trig_Select`
Set Trigger Delay	`MDAQ_Trig_Delay`
Set Scan Rate	`MDAQ_ScanRate`
Setup Multi-Channel DAQ	`MDAQ_Setup`
Start Multi-Channel DAQ	`MDAQ_Start`
Get Multi-Channel DAQ	`MDAQ_Get`
Get Multi-Channel DAQ to String	`MDAQ_StrGet`
Check Multi-Channel DAQ	`MDAQ_Check`

Table 1-3. The LabWindows/CVI Function Tree for Data Acquisition (Continued)

LabWindows/CVI Function Panel	NI-DAQ Function
Stop Multi-Channel DAQ	`MDAQ_Stop`
Clear Multi-Channel DAQ	`MDAQ_Clear`
Analog Output	
Single Point	
Generate Voltage	`AO_VWrite`
Scale Voltage to Binary	`AO_VScale`
Write Analog Binary	`AO_Write`
Update Analog DACs	`AO_Update`
Configure Analog Output	`AO_Configure`
Change Analog Output Parameter	`AO_Change_Parameter`
Waveform Generation	
Generate WFM from Array	`WFM_Op`
Generate WFM from Disk	`WFM_from_Disk`
Low-Level Functions	
Scale Waveform Buffer	`WFM_Scale`
Convert Waveform Rate	`WFM_Rate`
Assign Waveform Group	`WFM_Group_Setup`
Load Waveform Buffer	`WFM_Load`
Assign Rate to WFM Group	`WFM_ClockRate`
Control Waveform Group	`WFM_Group_Control`
Pause/Resume WFM Channel	`WFM_Chan_Control`
Check Waveform Channel	`WFM_Check`
Enable Double Buffering	`WFM_DB_Config`
Is Half Buffer Ready?	`WFM_DB_HalfReady`
Copy Array to WFM Buffer	`WFM_DB_Transfer`
Copy String to WFM Buffer	`WFM_DB_StrTransfer`
Digital Input/Output	
Configure Port	`DIG_Prt_Config`
Configure Line	`DIG_Line_Config`

Table 1-3. The LabWindows/CVI Function Tree for Data Acquisition (Continued)

LabWindows/CVI Function Panel	NI-DAQ Function
Read Port	`DIG_In_Port`
Read Line	`DIG_In_Line`
Write Port	`DIG_Out_Port`
Write Line	`DIG_Out_Line`
Get Port Status	`DIG_Prt_Status`
Group Mode	
Configure Group	`DIG_Grp_Config`
Read Group	`DIG_In_Grp`
Write Group	`DIG_Out_Grp`
Get Group Status	`DIG_Grp_Status`
Set Group Mode	`DIG_Grp_Mode`
Block Transfer	
Read Block	`DIG_Block_In`
Write Block	`DIG_Block_Out`
Check Block	`DIG_Block_Check`
Clear Block	`DIG_Block_Clear`
Set Up Pattern Generation	`DIG_Block_PG_Config`
Set Up Digital Scanning	`DIG_SCAN_Setup`
Enable Double Buffering	`DIG_DB_Config`
Is Half Buffer Ready?	`DIG_DB_HalfReady`
Transfer To/From Array	`DIG_DB_Transfer`
Transfer To/From String	`DIG_DB_StrTransfer`
SCXI	
Load SCXI Configuration	`SCXI_Load_Config`
Change Configuration	`SCXI_Set_Config`
Get Chassis Config Info	`SCXI_Get_Chassis_Info`
Get Module Config Info	`SCXI_Get_Module_Info`
Reset SCXI	`SCXI_Reset`
Set Up Single AI Channel	`SCXI_Single_Chan_Setup`

Table 1-3. The LabWindows/CVI Function Tree for Data Acquisition (Continued)

LabWindows/CVI Function Panel	NI-DAQ Function
Set Up Muxed Scanning	`SCXI_SCAN_Setup`
Set Up Mux Counter	`SCXI_MuxCtr_Setup`
Set Up Track/Hold	`SCXI_Track_Hold_Setup`
Control Track/Hold State	`SCXI_Track_Hold_Control`
Select Gain	`SCXI_Set_Gain`
Configure Filter	`SCXI_Configure_Filter`
Select Scanning Mode	`SCXI_Set_Input_Mode`
Change AI Channel	`SCXI_Change_Chan`
Scale SCXI Data	`SCXI_Scale`
Write to AO Channel	`SCXI_AO_Write`
Set Digital or Relay State	`SCXI_Set_State`
Get Digital or Relay State	`SCXI_Get_State`
Get Status Register	`SCXI_Get_Status`
Set Up Calibration Mode	`SCXI_Calibrate_Setup`
Change Cal Constants	`SCXI_Cal_Constants`
Counter/Timer	
DAQ-STC Counters (GPCTR)	
Select Ctr Application	`GPCTR_Set_Application`
Change Ctr Parameter	`GPCTR_Change_Parameter`
Configure Ctr Buffer	`GPCTR_Config_Buffer`
Control Ctr Operation	`GPCTR_Control`
Monitor Ctr Properties	`GPCTR_Watch`
Am9513 Counters (CTR)	
Configure Counter	`CTR_Config`
Count Events	`CTR_EvCount`
Count Periods	`CTR_Period`
Read Counter	`CTR_EvRead`
Stop Counter	`CTR_Stop`
Restart Counter	`CTR_Restart`
Reset Counter	`CTR_Reset`

Table 1-3. The LabWindows/CVI Function Tree for Data Acquisition (Continued)

LabWindows/CVI Function Panel	NI-DAQ Function
Get Counter Output State	`CTR_State`
Convert CTR Rate	`CTR_Rate`
Generate Pulse	`CTR_Pulse`
Generate Square Wave	`CTR_Square`
Generate Freq OUT Signal	`CTR_FOUT_Config`
Operate Multi Counters	`CTR_Simul_Op`
8253 Counters (ICTR)	
Setup Interval Counter	`ICTR_Setup`
Read Interval Counter	`ICTR_Read`
Reset Interval Counter	`ICTR_Reset`
RTSI Bus	
Connect RTSI	`RTSI_Conn`
Disconnect RTSI	`RTSI_DisConn`
Clear RTSI	`RTSI_Clear`
Clock RTSI	`RTSI_Clock`
Event Messaging	
Config Alarm Deadband	`Config_Alarm_Deadband`
Config Analog Trigger Event	`Config_ATrig_Event_Message`
Config Event Message	`Config_DAQ_Event_Message`
Get an Event Message	`Get_DAQ_Event`
Peek at an Event Message	`Peek_DAQ_Event`

Initialization/Utilities is a class of functions used for general board initialization and configuration, for configuration retrieval, and for setting NI-DAQ properties. This class also contains several useful utility functions.

Board Config & Calibrate is a class of functions that perform calibration and configuration that is specific to a single type of board.

The *Analog Input* class contains all of the classes of functions that perform A/D conversions.

Single Point is a class of Analog Input functions that perform A/D conversions of a single sample.

Multiple Point is a class of functions that perform clocked, buffered multiple A/D conversions typically used to capture waveforms. This class includes high-level functions and a *Low-Level Functions* subclass. The high-level functions are synchronous; that is, your application is blocked while these functions are performing the requested number of A/D conversions. The low-level functions are asynchronous; that is, your application continues to run while the board performs A/D conversions in the background. The low-level functions also include the double-buffered functions.

Simultaneous Sample/Hold is a class containing two subclasses of functions that work only on boards with a simultaneous sample-and-hold architecture (currently the EISA-A2000, the AT-A2150, and the AT-DSP2200). The *Single-Scan (MAI***)** subclass contains functions that acquire just one scan of analog input data. The *Multiple-Scan (MDAQ)* subclass contains functions that perform clocked, buffered multiple-scan A/D conversions.

The *Analog Output* class contains all the classes of functions that perform D/A conversions.

Single Point is a class of Analog Output functions that perform single D/A conversions.

Waveform Generation is a class of functions that perform buffered analog output. The Waveform Generation functions generate waveforms from data contained in an array or a disk file. The *Low-Level Functions* subclass provides a finer level of control in generating multiple D/A conversions.

Digital Input/Output is a class of functions that perform digital input and output operations. It also contains two subclasses. *Group Mode* is a subclass of the *Digital Input/Output* class that contains functions for handshaked digital input and output operations. *Block Transfer* is a subclass of the *Group Mode* class that contains functions for handshaked or clocked, buffered or double-buffered digital input and output operations.

SCXI is a class of functions used to configure the SCXI line of signal conditioning products.

Counter/Timer is a class of function panels that perform counting and timing operations. *DAQ-STC Counters (GPCTR)* is a subclass of Counter/Timer that contains functions that perform operations on the DAQ-STC counters on the E Series devices. *Am9513 Counters (CTR)* is another subclass of Counter/Timer that contains functions that perform operations on the Am9513 counters on the Am9513-based devices, the EISA-A2000, and the PC-TIO-10. *8253 Counters (ICTR)* is a subclass of Counter/Timer that contains functions that perform counting and timing operations for the DAQCard-500/700 and 516, Lab and 1200 series, and LPM devices.

RTSI Bus is a class of function panels that connect control signals to the RTSI bus and to other boards.

The *DAQ Event Messages* class contains functions that set up conditions for sending messages to your application when certain events occur.

Device Numbers and SCXI Chassis IDs

The first parameter to almost every NI-DAQ function is the device number of the DAQ device you want NI-DAQ to use for the given operation. After you have followed the installation and configuration instructions in the NI-DAQ release notes and Chapter 1, *Introduction to NI-DAQ*, of the *NI-DAQ User Manual for PC Compatibles*, the configuration utility displays the device number for each device you have installed in the system. You can use the configuration utility to verify your device numbers. You can use multiple DAQ devices in one application; to do so, simply pass the appropriate device number to each function.

If you are using SCXI, you must pass the chassis ID that you assigned to your SCXI chassis in the configuration utility to the SCXI functions that you use. For many of the SCXI functions, you must also pass the module slot number of the module you want to use. The slots in the SCXI chassis are numbered from left to right, beginning with slot 1. The controller on the left side of the chassis is referred to as Slot 0. You can use the configuration utility to verify your chassis IDs and your module slot numbers.

Chapter 2

Function Reference

This chapter contains a detailed explanation of each NI-DAQ function. The functions are arranged alphabetically.

A2000_Calibrate

Format

status = A2000_Calibrate (deviceNumber, saveNewValues, calMethod, channel, extRefVoltage)

Purpose

Calibrates the EISA-A2000 A/D gain and offset values or restores them to the original factory-set values. You will use the gain and offset values calculated during calibration to adjust the accuracy of the readings from the four analog input channels. Notice that NI-DAQ automatically loads the stored calibration values the first time you call a function pertaining to the EISA-A2000.

Warning: ***Read the calibration chapter in the*** *EISA-A2000 User Manual* ***before using*** `A2000_Calibrate`**.**

A2000_Calibrate

Continued

Parameters

Input

Name	Type		Description
	Windows	Windows NT	
deviceNumber	I16	I32	assigned by configuration utility
saveNewValues	I16	I32	selects whether to store the values used in the calibration
calMethod	I16	I32	calibration method to use
channel	I16	I32	input channel connected to the external reference source
extRefVoltage	F64	F64	voltage of the external reference

Parameter Discussion

saveNewValues selects whether to store the values used in the calibration. The gain and offset calibration values are stored in an EEPROM on the EISA-A2000 board, which does not lose its data even when there is no power to the board. NI-DAQ reads these values from the EEPROM and loads these values into the EISA-A2000 calibration circuitry when the board is initialized (by `Init_DA_Brds` called directly or indirectly when a NI-DAQ function is first called) or reset (`Init_DA_Brds`) and saved for use during data acquisition. When you calibrate the EISA-A2000, you can choose to replace the permanent copies of the gain and offset values in the EEPROM and use the new values until the next calibration, even if the board is reinitialized, or you can elect not to replace the EEPROM values but use the new values until the next calibration or initialization. Set **saveNewValues** as follows:

0: Do not write new values to EEPROM.
1: Write new values to EEPROM.

A2000_Calibrate

Continued

For example, if you get consistently inaccurate readings from one or more input channels, even after resetting the board, you can calibrate and save the new gain and offset calibration values as permanent copies in the EEPROM. However, if acquisition results are accurate after initialization but start to drift after a few hours of operation when the temperature of the board has increased, you can calibrate the board at this operating temperature but retain the current EEPROM values to use after the next initialization.

calMethod indicates which calibration method to use.

0:	Use internal reference to calibrate.
1:	Use external reference to calibrate.
2:	Reload factory calibration values.

channel indicates which input channel is connected to the external reference source. For greatest accuracy in the calibration, connect the reference to more than one channel, and set **channel** to -1. NI-DAQ will average all channels with input values close to the given **extRefVoltage** to find the reference voltage. If you have the reference voltage connected to only one input channel, set the channel to that channel number.

-1:	Source channels are determined automatically and values averaged.
0:	Channel 0.
1:	Channel 1.
2:	Channel 2.
3:	Channel 3.

extRefVoltage is the voltage of the external reference.
Range: +3 to +5 V or -5 to -3 V.

A2000_Config

Format

status = A2000_Config (deviceNumber, sampClkSource, sampClkDrive, dither)

Purpose

Configures some special EISA-A2000 features, such as selecting the source of the sample clock, whether to drive the SAMPCLK* line, and whether to add dithering to the input signal.

Parameters

Input

Name	Type		Description
	Windows	Windows NT	
deviceNumber	I16	I32	assigned by configuration utility
sampClkSource	I16	I32	onboard sample clock or external signal
sampClkDrive	I16	I32	whether to drive SAMPCLK* line
dither	I16	I32	whether to add white Gaussian noise to the input signal

Parameter Discussion

sampClkSource indicates whether timing of sampling during data acquisition is controlled by the onboard sample clock or by an external signal.

0: Onboard sample clock.
1: External sample clock.

When **sampClkSource** is set to 1, the EISA-A2000 receives the sample clock from the external SAMPCLK* line on the I/O connector or from the CLOCKI line of the RTSI bus. To receive the sample clock from the RTSI CLOCKI line, call `RTSI_Conn`.

A2000_Config

Continued

sampClkDrive indicates whether to drive the sample clock signal onto the SAMPCLK* line.

0:	Sample clock signal does not drive SAMPCLK* line.
1:	Sample clock signal drives SAMPCLK* line.

It is not possible to receive the sample clock from the SAMPCLK* line and drive it at the same time.

dither indicates whether to add approximately 0.5 LSB rms of white Gaussian noise to the input signal. This is useful for applications that involve averaging to increase the resolution of the EISA-A2000 to more than 12 bits. For high-speed applications that do not involve averaging, dithering is not recommended and should be disabled.

0:	Dither disabled.
1:	Dither enabled.

Using This Function

After system startup, the EISA-A2000 is configured for the following:

- **sampClkSource** = 0: Onboard sample clock.
- **sampClkDrive** = 0: Sample clock signal does not drive SAMPCLK* line.
- **dither** = 0: Dither disabled.

As mentioned in the description of **sampClkDrive**, it is not possible to receive the sample clock signal from the SAMPCLK* line and drive the SAMPCLK* line simultaneously. However, because setting **sampClkSource** to 1 indicates that the board receives the sample clock from an external source that can be either the SAMPCLK* line or the CLOCKI signal of the RTSI bus, NI-DAQ does not detect conflicts until you call `MAI_Arm` or `MDAQ_Start` and the source of the external sample clock is known.

A2150_Calibrate

Format

status = A2150_Calibrate (deviceNumber, ref0, ref1)

Purpose

Performs offset calibrations on the ADCs of the specified AT-A2150.

Parameters

Input

Name	Type		Description
	Windows	Windows NT	
deviceNumber	I16	I32	assigned by configuration utility
ref0	I16	I32	reference source used to calibrate ADC0
ref1	I16	I32	reference source used to calibrate ADC1

Parameter Discussion

ref0 specifies the reference source used to calibrate ADC0 on the AT-A2150.

0: Use the analog input ground as the reference.
1: Use the external signals connected to ACH0 and ACH1 as ground references.

ref1 specifies the reference source used to calibrate ADC1 on the AT-A2150.

0: Use the analog input ground as the reference.
1: Use the external signals connected to ACH2 and ACH3 as ground references.

Using This Function

When `Init_DA_Brds` initializes the AT-A2150, an offset calibration is performed using the analog ground as the reference. This function is necessary only if you want to either calibrate to an external reference or recalibrate the board for ground reference after using an external reference.

AI_Check

Format

status = AI_Check (deviceNumber, readingAvailable, reading)

Purpose

Returns the status of the analog input circuitry and an analog input reading if one is available. `AI_Check` is intended for use when A/D conversions are initiated by external pulses applied at the EXTCONV* pin or, if you are using the E Series devices, at the pin selected through the `Select_Signal` function; see `DAQ_Config` for information on enabling external conversions.

Parameters

Input

Name	Type		Description
	Windows	Windows NT	
deviceNumber	I16	I32	assigned by configuration utility

Output

Name	Type		Description
	Windows	Windows NT	
readingAvailable	I16	I32	whether a reading is available
reading	I16	I16	integer result

Parameter Discussion

readingAvailable represents the status of the analog input circuitry.

1:	NI-DAQ returns an A/D conversion result in **reading**.
0:	No A/D conversion result is available.

AI_Check

Continued

reading is the integer in which NI-DAQ returns the result of an A/D conversion. If the device is configured for unipolar operation, **reading** ranges from 0 to 4,095. If the device is configured for bipolar operation, **reading** ranges from -2,048 to +2,047. For devices with 16-bit ADCs, **reading** ranges from 0 to 65,535 in unipolar operation, and -32,768 to +32,767 in bipolar operation.

☞ **Note:** ***C Programmers*—readingAvailable *and* reading *are pass-by-reference parameters.***

Using This Function

`AI_Check` checks the status of the analog input circuitry. If the device has performed an A/D conversion, `AI_Check` returns **readingAvailable** = 1 and the A/D conversion result. Otherwise, `AI_Check` returns **readingAvailable** = 0.

`AI_Setup`, in conjunction with `AI_Check` and `AI_Clear`, is useful for externally timed A/D conversions. After you call `AI_Setup`, you can call `AI_Clear` to clear out the A/D FIFO of any previous conversion results. The device then performs a conversion each time the device receives a pulse at the appropriate pin. You can call `AI_Check` to check for and return available conversion results.

You cannot use this function if you have an SC-2040 connected to your DAQ device.

AI_Clear

Format

status = AI_Clear (deviceNumber)

Purpose

Clears the analog input circuitry and empties the FIFO memory.

Parameters

Input

Name	Type		Description
	Windows	**Windows NT**	
deviceNumber	I16	I32	assigned by configuration utility

Using This Function

`AI_Clear` clears the analog input circuitry and empties the analog input FIFO memory. `AI_Clear` also clears any analog input error conditions. You should call `AI_Clear` to clear out the A/D FIFO memory before any series of externally triggered conversions begins.

AI_Configure

Format

status = AI_Configure (deviceNumber, chan, inputMode, inputRange, polarity, driveAIS)

Purpose

Informs NI-DAQ of the input mode (single-ended or differential), input range, and input polarity selected for the device. Use this function if you have changed the jumpers affecting the analog input configuration from their factory settings. For devices that have no jumpers for analog input configuration, this function programs the device for the settings you want.

If you are using NI-DAQ for Windows and you have recorded a nondefault analog input configuration through the NI-DAQ Configuration Utility, you do not need to use `AI_Configure` because NI-DAQ uses the settings recorded by the NI-DAQ Configuration Utility. If you have a software-configurable device, you can use `AI_Configure` to change the analog input configuration on the fly.

For the E Series devices, AT-MIO-64F-5, and AT-MIO-16X, you can configure the input mode and polarity on a per-channel basis. For the AT-MIO-64F-5 and the AT-MIO-16X, you can use `AI_Configure` to specify whether to drive AISENSE to onboard ground.

Parameters

Input

Name	Type		Description
	Windows	Windows NT	
deviceNumber	I16	I32	assigned by configuration utility
chan	I16	I32	channel to be configured
inputMode	I16	I32	indicates whether channels are configured for single-ended or differential operation

AI_Configure

Continued

Name	**Type**		**Description**
	Windows	**Windows NT**	
inputRange	I16	I32	voltage range of the analog input channels
polarity	I16	I32	indicates whether the ADC is configured for unipolar or bipolar operation
driveAIS	I16	I32	indicates whether to drive AISENSE to onboard ground

Parameter Discussion

chan is the analog input channel to be configured. Except for the E Series devices, the AT-MIO-64F-5, and the AT-MIO-16X, you must set **chan** to -1 because the same analog input configuration applies to all of the channels. For the E Series devices, AT-MIO-64F-5, and AT-MIO-16X, **chan** specifies the channel to be configured. If you want all of the channels to be configured identically, set **chan** to -1. For ranges, see Table B-1 in Appendix B.

inputMode indicates whether the analog input channels are configured for single-ended or differential operation:

0: Differential (DIFF) configuration (default).

1: Referenced Single-Ended (RSE) configuration (used when the input signal does not have its own ground reference. The negative (-) input of the instrumentation amplifier is tied to the instrumentation amplifier signal ground to provide one).

2: Nonreferenced Single-Ended (NRSE) configuration (used when the input signal has its own ground reference. The input signal’s ground reference is connected to AISENSE, which is tied to the negative (-) input of the instrumentation amplifier).

inputRange is the voltage range of the analog input channels.

AI_Configure

Continued

polarity indicates whether the ADC is configured for unipolar or bipolar operation:

0:	Bipolar operation (default value).
1:	Unipolar operation.

The following table shows all possible settings for **inputMode**, **inputRange**, and **polarity**, with the default settings in italics. **inputMode** is independent of **inputRange** and **polarity**.

Device	**Possible Values for inputMode***	**Analog Input Range**			**Software Configurable**
		inputRange*	**polarity***	**Resulting Analog Input Range**	
AT-MIO-64F-5, AT-MIO-16F-5, 12-bit E Series	*0*, 1, 2	ignored	unipolar	0 to +10 V	yes
		ignored	*bipolar*	-5 to +5 V	
AT-MIO-16X, 16-bit E Series	*0*, 1, 2	ignored	unipolar	0 to +10 V	yes
		ignored	*bipolar*	-10 to +10 V	
MIO-16 and AT-MIO-16D	*0*, 1, 2	10	unipolar	0 to +10 V	no
		10	bipolar	-5 to +5 V	
		20	*bipolar*	-10 to +10 V	
Lab-PC+	0, *1*	ignored	unipolar	0 to +10 V	no
		ignored	*bipolar*	-5 to +5 V	
DAQCard-1200, DAQPad-1200, Lab-PC-1200, Lab-PC-1200AI, SCXI-1200	0, *1*	ignored	unipolar	0 to +10 V	yes
		ignored	*bipolar*	-5 to +5 V	

* Italics indicates default settings.

AI_Configure

Continued

Device	**Possible Values for inputMode***	**Analog Input Range**			**Software Configurable**
		inputRange*	**polarity***	**Resulting Analog Input Range**	
LPM Devices (RSE **inputMode** only)	ignored	5	unipolar	0 to +5 V	no (PC-LPM-16)
		5	bipolar	-2.5 to +2.5 V	yes (PC-LPM-16PnP)
		10	unipolar	0 to +10 V	
		10	*bipolar*	-5 to +5 V	
516 Devices, DAQCard-500	*1*	*10*	*bipolar*	-5 to 5 V	n/a
DAQCard-700	0, *1*	5	bipolar	-2.5 to +2.5 V	yes
		10	bipolar	-5 to +5 V	
		20	bipolar	-10 to +10 V	
* Italics indicates default settings.					

☞ **Note:** ***If a device is software configurable, the*** **inputMode***,* **inputRange***, and* **polarity** ***parameters are used to program the device for the configuration you want. If a device is not software configurable, this function uses these parameters to inform NI-DAQ of the device configuration, which you must set using hardware jumpers. If your device is software configurable and you have changed the analog input settings through the NI-DAQ Configuration Utility, you do not have to use*** `AI_Configure`***, although it is good practice to do so in case you inadvertently change the configuration file maintained by the NI-DAQ Configuration Utility.***

AI_Configure

Continued

driveAIS, for the AT-MIO-64F-5 and AT-MIO-16X, indicates whether to drive AISENSE to onboard ground or not. This parameter is ignored for all other devices.

0:	Do not drive AISENSE to ground.
1:	Drive AISENSE to ground.

Notice that if you have configured any of the input channels in nonreferenced single-ended (NRSE) mode, this function returns a warning, **inputModeConflict** (18), if you set **driveAIS** to 1. When NI-DAQ reads a channel in NRSE mode, the device uses AISENSE as an input to the negative input of the amplifier, regardless of the **driveAIS** setting. When NI-DAQ reads a channel in differential or referenced single-ended (RSE) mode, the device drives AISENSE to onboard ground if **driveAIS** is 1.

Using This Function

When you attach an SC-2040 or SC-2042-RTD to your DAQ device, you must configure channels 0 through 7 for differential mode. When you attach an SC-2043-SG to your DAQ device, you must configure these channels for nonreferenced single-ended mode.

On the AT-MIO-16X, 16-bit E Series, AT-MIO-16F-5, and AT-MIO-64F-5 devices, the calibration constants used for analog input change depending on the polarity of the analog input channels. NI-DAQ always ensures that the calibration constants in use match the current polarity of the channels.

See the `Calibrate_E_Series` function description for information about calibration constant loading on the E Series devices.

If you change the polarity on AT-MIO-16X, AT-MIO-64F, and AT-MIO-16F-5 by calling `AI_Configure`, NI-DAQ uses the following guidelines to ensure that appropriate constants are loaded *automatically*.

- AT-MIO-16X: NI-DAQ checks if the load area contains the appropriate constants. If so, NI-DAQ will load the constants from the load area. Otherwise, NI-DAQ will load the constants from the factory area for the current polarity and return status code **calConstPolarityConflictError**.
- AT-MIO-64F-5: This device has separate caldacs for unipolar and bipolar input. Therefore, NI-DAQ does not need to reload calibration constants.
- AT-MIO-16F-5: The load area on this device contains constants for both unipolar and bipolar input. Therefore, NI-DAQ will load the appropriate constants from the load area for the current polarity.

AI_Configure

Continued

☞ **Note:** ***The actual loading of calibration constants will take place when you call an*** `AI`***,*** `DAQ`***, or*** `SCAN` ***function. On the AT-MIO-16X, the need for reloading the constants will depend on the polarity of the channel on which you are doing analog input.***

AI_Mux_Config

Format

status = AI_Mux_Config (deviceNumber, numMuxBrds)

Purpose

Configures the number of multiplexer (AMUX-64T) devices connected to the MIO and AI devices and informs NI-DAQ of the presence of any AMUX-64T devices attached to the system (MIO and AI devices only).

Parameters

Input

Name	Type		Description
	Windows	Windows NT	
deviceNumber	I16	I32	assigned by configuration utility
numMuxBrds	I16	I32	number of external multiplexer devices

Parameter Discussion

numMuxBrds is the number of external multiplexer devices connected.

0:	No external AMUX-64T devices are connected (default).
1, 2, 4:	Number of AMUX-64T devices connected.

Using This Function

You use an external multiplexer device (AMUX-64T) to expand the number of analog input signals that you can measure with the MIO and AI device. The AMUX-64T has 16 separate four-to-one analog multiplexer circuits. One AMUX-64T reads 64 single-ended (32 differential) analog input signals. You can cascade four AMUX-64T devices to permit up to 256 single-ended (128 differential) analog input signals to be read through one MIO or AI device. See Chapter 13, *AMUX-64T External Multiplexer Device*, in the *DAQ Hardware Overview Guide* for more information on using the AMUX-64T.

AI_Mux_Config

Continued

`AI_Mux_Config` configures the number of multiplexer devices connected to the MIO or AI device. Input channels are then referenced in subsequent analog input calls (`AI_VRead`, `AI_Read`, `AI_Setup`, and `DAQ_Start`, for example) with respect to the external AMUX-64T analog input channels, instead of the MIO and AI device onboard channel numbers. You need to execute the call to `AI_Mux_Config` only once in an application program.

For the AT-MIO-64F-5 or AT-MIO-64E-3, you must also call `MIO_Config` if you plan to use AMUX-64T channels. Refer to the `MIO_Config` call for further details.

☞ **Note:** ***Some of the digital lines of port 0 on the MIO or AI device with AMUX-64T devices are reserved for AMUX device control. Any attempt to change the port or line direction or the digital values of the reserved line will cause an error. Table 2-1 shows the relationship between the number of AMUX-64T devices assigned to the MIO or AI device and the number of digital I/O lines reserved. You can use the remaining lines of port 0. On non-E Series devices, the remaining lines are available for output only.***

Table 2-1. Port 0 Digital I/O Lines Reserved

Number of AMUX-64T Devices Assigned to MIO or AI Device	Port 0 Digital Lines Reserved
0	None
1	Lines 0 and 1
2	Lines 0, 1, and 2
4	Lines 0, 1, 2, and 3

AI_Read

Format

status = AI_Read (deviceNumber, chan, gain, reading)

Purpose

Reads an analog input channel (initiates an A/D conversion on an analog input channel) and returns the unscaled result.

Parameters

Input

Name	Type		Description
	Windows	Windows NT	
deviceNumber	I16	I32	assigned by configuration utility
chan	I16	I32	analog input channel number
gain	I16	I32	gain setting for the channel

Output

Name	Type		Description
	Windows	Windows NT	
reading	I16	I16	the integer result of the A/D conversion

Parameter Discussion

chan is the analog input channel number. If you are using SCXI, you must use the appropriate analog input channel on the DAQ device that corresponds to the SCXI channel you want. To select the SCXI channel, use `SCXI_Single_Chan_Setup` before calling this function. Please refer to Chapter 15, *SCXI Hardware,* in the *DAQ*

AI_Read

Continued

Hardware Overview Guide and the *NI-DAQ User Manual for PC Compatibles* for more information on SCXI channel assignments.

For ranges see Table B-1 in Appendix B.

gain is the gain setting you will use for the specified channel. This gain setting applies only to the DAQ device; if you are using SCXI, establish any gain you want on the SCXI module either by setting jumpers on the module or by calling `SCXI_Set_Gain`. Refer to Appendix B, *Analog Input Channel and Gain Settings and Voltage Calculation*, for valid gain settings. If you use an invalid gain, NI-DAQ returns an error. If you call `AI_Read` for the DAQCard-500/700 or 516 and LPM devices, NI-DAQ ignores the gain.

Note: ***NI-DAQ does not distinguish between the low-gain and high-gain versions of the AT-MIO-16. If you enter a gain of 10 and you have a device with gains of 1, 2, 4, and 8, then a gain of 2 is actually used and no error is returned.***

reading is the integer in which NI-DAQ returns the 12-bit or 16-bit result of the A/D conversion.

Range:	0 to 4,095 (12-bit devices, unipolar mode).
	-2,048 to 2,047 (12-bit devices, bipolar mode).
	0 to 65,535 (16-bit devices, unipolar mode).
	-32,768 to 32,767 (16-bit devices, bipolar mode).

Note: ***C Programmers*—reading *is a pass-by-reference parameter.***

Using This Function

`AI_Read` addresses the specified analog input channel, changes the input gain to the specified gain setting, and initiates an A/D conversion. `AI_Read` waits for the conversion to complete and returns the result. If the conversion does not complete within a reasonable time, the call to `AI_Read` is said to have *timed out* and the **timeOutError** code is returned.

AI_Read_Scan

Format

status = AI_Read_Scan (deviceNumber, reading)

Purpose

Returns readings for all analog input channels selected by `SCAN_Setup` (E Series devices only, with or without the SC-2040 accessory).

Parameters

Input

Name	Type		Description
	Windows	Windows NT	
deviceNumber	I16	n/a	assigned by configuration utility

Output

Name	Type		Description
	Windows	Windows NT	
reading	[I16]	n/a	readings from each sampled analog input channel

Parameter Discussion

reading is an array of readings from each sampled analog input channel. The length of the **reading** array is equal to the number of channels selected in the `SCAN_Setup` **numChans** parameter. Range of elements in **reading** depends on your device A/D converter resolution and the unipolar/bipolar selection you make for a given channel.

Using This Function

`AI_Read_Scan` samples the analog input channels selected by `SCAN_Setup` at half the maximum rate permitted by your hardware.

AI_Setup

Format

status = AI_Setup (deviceNumber, chan, gain)

Purpose

Selects the specified analog input channel and gain setting for externally pulsed conversion operations.

Parameters

Input

Name	Type		Description
	Windows	Windows NT	
deviceNumber	I16	I32	assigned by configuration utility
chan	I16	I32	analog input channel number
gain	I16	I32	gain setting to be used

Parameter Discussion

chan is the analog input channel number. If you are using SCXI, you must use the appropriate analog input channel on the DAQ device that corresponds to the SCXI channel you want. To select the SCXI channel, use `SCXI_Single_Chan_Setup` before calling this function. Please refer to Chapter 15, *SCXI Hardware,* in the *DAQ Hardware Overview Guide* and the *NI-DAQ User Manual for PC Compatibles* for more information on SCXI channel assignments.

Range: See Table B-1 in the *DAQ Device Analog Input Channel Settings* section of Appendix B, *Analog Input Channel and Gain Settings and Voltage Calculation*.

gain is the gain setting to be used for the specified channel. **gain** applies only to the DAQ device; if you are using SCXI, establish any gain you want on the SCXI module by setting jumpers on the module (if any) or by calling `SCXI_Set_Gain`. Refer to Appendix B, *Analog Input Channel and Gain Settings and Voltage Calculation*, for valid gain settings. If you use an invalid gain, NI-DAQ returns an error. If you call `AI_Setup` for the 516 and LPM devices or DAQCard-500/700, NI-DAQ ignores the gain.

AI_Setup

Continued

☞ **Note:** ***NI-DAQ does not distinguish between the low-gain and high-gain versions of the AT-MIO-16. If you enter a gain of 10 and you have a device with gains of 1, 2, 4, and 8, then NI-DAQ uses a gain of 2 and returns no error.***

Using This Function

`AI_Setup` addresses the specified analog input channel and changes the input gain to the specified gain setting. `AI_Setup`, in conjunction with `AI_Check` and `AI_Clear`, is useful for externally timed A/D conversions. If your application calls `AI_Read` with channel and gain parameters different from those used in the last `AI_Setup` call, you must call `AI_Setup` again for `AI_Check` to return data from the channel you want at the selected gain.

This function cannot be used if you have an SC-2040 connected to your DAQ device.

AI_VRead

Format

status = AI_VRead (deviceNumber, chan, gain, voltage)

Purpose

Reads an analog input channel (initiates an A/D conversion on an analog input channel) and returns the result scaled to a voltage in units of volts.

Parameters

Input

Name	Type		Description
	Windows	Windows NT	
deviceNumber	I16	I32	assigned by configuration utility
chan	I16	I32	analog input channel number
gain	I16	I32	gain setting to be used for the specified channel

Output

Name	Type		Description
	Windows	Windows NT	
voltage	F64	F64	the measured voltage returned, scaled to units of volts

Parameter Discussion

chan is the analog input channel number.
Range: See Table B-1 in Appendix B.

AI_VRead

Continued

gain is the gain setting to be used for the specified channel. Refer to Appendix B, *Analog Input Channel and Gain Settings and Voltage Calculation*, for valid gain settings. If you use an invalid gain, NI-DAQ returns an error. If you call `AI_VRead` for the 516 and LPM devices or DAQCard-500/700, NI-DAQ ignores the gain.

☞ **Note:** ***NI-DAQ does not distinguish between the low-gain and high-gain versions of the AT-MIO-16. If you enter a gain of 10 and you have a device with gains of 1, 2, 4, and 8, then NI-DAQ uses a gain of 2 and returns no error.***

voltage is the floating-point variable in which NI-DAQ returns the measured voltage, scaled to units of volts.

☞ **Note:** ***C Programmers—*voltage *is a pass-by-reference parameter.***

Using This Function

`AI_VRead` addresses the specified analog input channel, changes the input gain to the specified gain setting, and initiates an A/D conversion. `AI_VRead` waits for the conversion to complete and then scales and returns the result. If the conversion does not complete within a reasonable time, the call to `AI_VRead` is said to have *timed out* and NI-DAQ returns the **timeOutError** code.

When you use SCXI as a front end for analog input to an MIO or AI device, Lab-PC+, Lab-PC-1200, Lab-PC-1200AI, LPM device, or DAQCard-700, it is not advisable to use the `AI_VRead` function because that function does not take into account the gain of the SCXI module when scaling the data. You should use the `AI_Read` function to get unscaled data and then call the `SCXI_Scale` function.

When you have an SC-2040 accessory connected to an E Series device, this function takes both the onboard gains and the gains on SC-2040 into account while scaling the data. When you have an SC-2043-SG accessory connected to your DAQ device, this function takes both the onboard gains and the SC-2043-SG fixed gain of 10 into account while scaling the data.

AI_VRead_Scan

Format

status = AI_VRead_Scan (deviceNumber, reading)

Purpose

Returns readings in volts for all analog input channels selected by `SCAN_Setup` (E Series devices only with or without the SC-2040 accessory).

Parameters

Input

Name	Type		Description
	Windows	Windows NT	
deviceNumber	I16	n/a	assigned by configuration utility

Output

Name	Type		Description
	Windows	Windows NT	
reading	[F64]	n/a	voltage readings from each sampled analog input channel

Parameter Discussion

reading is an array of readings from each sampled analog input channel. The length of the **reading** array is equal to the number of channels selected in the `SCAN_Setup` **numChans** parameter. NI-DAQ uses values you have specified in `SCAN_Setup` through the **gains** parameter for computing voltages. If you have attached an SC-2040 or SC-2043-SG to your DAQ device, NI-DAQ also uses values you have specified in `SC_2040_Config` (through the **sc2040gain** parameter) or `Set_DAQ_Device_Info` (a fixed gain of 10) for computing voltages.

AI_VRead_Scan

Continued

Using This Function

`AI_VRead_Scan` samples the analog input channels selected by `SCAN_Setup` at half the maximum rate of your data acquisition hardware.

You must use the `SCAN_Setup` function prior to invoking this function.

You cannot use external signals to control A/D conversion timing and use this function at the same time.

AI_VScale

Format

status = AI_VScale (deviceNumber, chan, gain, gainAdjust, offset, reading, voltage)

Purpose

Converts the binary result from an `AI_Read` call to the actual input voltage.

Parameters

Input

Name	Type		Description
	Windows	Windows NT	
deviceNumber	I16	I32	assigned by configuration utility
chan	I16	I32	channel on which binary reading was taken
gain	I16	I32	gain setting used to take the reading
gainAdjust	F64	F64	multiplying factor to adjust gain
offset	F64	F64	binary offset present in reading
reading	I16	I16	result of the A/D conversion

Output

Name	Type		Description
	Windows	Windows NT	
voltage	F64	F64	computed floating-point voltage

AI_VScale

Continued

Parameter Discussion

chan is the onboard channel or AMUX channel on which NI-DAQ took the binary reading using `AI_Read`. For devices other than the AT-MIO-16X, AT-MIO-64F-5, and E Series devices, this parameter is ignored because the scaling calculation is the same for all of the channels. However, you are encouraged to pass the correct channel number.

gain is the gain setting that you used to take the analog input reading. If you used SCXI to take the reading, this gain parameter should be the product of the gain on the SCXI module channel and the gain that the DAQ device used. Refer to Appendix B, *Analog Input Channel and Gain Settings and Voltage Calculation*, for valid gain settings. Use of invalid gain settings causes NI-DAQ to return an error unless you are using SCXI. If you call `AI_VScale` for the 516 and LPM devices or DAQCard-500/700, NI-DAQ ignores the gain unless you are using SCXI.

gainAdjust is the multiplying factor to adjust the gain. Refer to Appendix B, *Analog Input Channel and Gain Settings and Voltage Calculation*, for the procedure for determining **gainAdjust**. If you do not want to do any gain adjustment—for example, use the ideal gain as specified by the **gain** parameter—set **gainAdjust** to 1.

offset is the binary offset that needs to be subtracted from the **reading**. Refer to Appendix B, *Analog Input Channel and Gain Settings and Voltage Calculation*, for the procedure for determining offset. If you do not want to do any offset compensation, set **offset** to 0.

reading is the result of the A/D conversion returned by `AI_Read`.

voltage is the variable in which NI-DAQ returns the input voltage converted from **reading**.

Note: ***C Programmers*—voltage *is a pass-by-reference parameter.***

Using This Function

Refer to Appendix B, *Analog Input Channel and Gain Settings and Voltage Calculation*, for the formula `AI_VScale` uses to calculate **voltage** from **reading**.

If your device polarity and range settings differ from the default settings shown in the `Init_DA_Brds` function, be sure to call `AI_Configure` to inform the driver of the correct polarity and range before using this function.

You must use the `SCAN_Setup` function prior to invoking this function.

AI_VScale

Continued

You cannot use external signals to control A/D conversion timing and use this function at the same time.

Align_DMA_Buffer

Format

status = Align_DMA_Buffer (deviceNumber, resource, buffer, count, bufferSize, alignIndex)

Purpose

Aligns the data in a DMA buffer to avoid crossing a physical page boundary. This function is for use with DMA waveform generation and digital I/O pattern generation (AT-MIO-16F-5 and AT-DIO-32F only).

Parameters

Input

Name	**Type**		**Description**
	Windows	**Windows NT**	
deviceNumber	I16	I32	assigned by configuration utility
resource	I16	I32	represents the DAC channel or the digital input or output group
buffer	[I16], HDL	[I16], HDL	integer array of samples to be used
count	U32	U32	number of data samples
bufferSize	U32	U32	actual size of **buffer**

Output

Name	**Type**		**Description**
	Windows	**Windows NT**	
alignIndex	U32	U32	offset into the array of the first data sample

Align_DMA_Buffer

Continued

Parameter Discussion

resource represents the DAC channel (for waveform generation) or the digital input or output group (for pattern generation) for which NI-DAQ will use the buffer.

0:	DAC channel 0.
1:	DAC channel 1.
2:	DAC channels 0 and 1.
11:	DIG group 1 (group size of 2).
12:	DIG group 2 (group size of 2).
13:	DIG group 1 (group size of 4).

buffer is the integer array of samples NI-DAQ will use in the waveform or pattern generation. The actual size of **buffer** should be larger than the number of samples to make room for possible alignment. If the actual size of the buffer is not big enough for alignment, the function returns a **memAlignmentError**. For Windows applications running in real or standard mode, a **bufferSize** of 2∗**count** guarantees that there is enough room for alignment.

count is the number of data samples contained in **buffer**.
Range: 3 through 2^{32}-1.

bufferSize is the actual size of **buffer**.
Range: **count** through 2^{32}-1.

alignIndex is the variable in which NI-DAQ returns the offset into the array of the first data sample. If NI-DAQ did not have to align the buffer, NI-DAQ returns **alignIndex** as 0, indicating that the data is still located at the beginning of the buffer. If NI-DAQ aligned the buffer to avoid a page boundary, **alignIndex** is a value other than 0, and the first data sample is located at **buffer**[**alignIndex**] (if your array is zero based). If you use digital input with an aligned buffer, NI-DAQ stores the data in the buffer beginning at **alignIndex**.

☞ **Note:** ***C Programmers*—alignIndex *is a pass-by-reference parameter.***

Using This Function

Use `Align_DMA_Buffer` to avoid the negative effects of page boundaries in the data buffer on AT bus machines for the following cases:

- DMA waveform generation at close to maximum speed
- Digital I/O pattern generation at close to maximum speed

Align_DMA_Buffer

Continued

- Interleaved DMA waveform generation at any speed
- 32-bit digital I/O pattern generation at any speed

The possibility of a page boundary occurring in the data buffer increases with the size of the buffer. When a page boundary occurs in the data buffer, NI-DAQ must reprogram the DMA controller before NI-DAQ can transfer the next data sample. The extra time needed to do the reprogramming increases the minimum update interval (thus decreasing the maximum update rate).

A page boundary in an interleaved DMA waveform buffer or a buffer that is to be used for 32-bit digital pattern generation can cause unpredictable results, *regardless of your operating speed*. To avoid this problem, you should *always* use `Align_DMA_Buffer` with interleaved DMA waveform generation (indicated by **resource** = 2) and 32-bit digital pattern generation (indicated by **resource** = 13). In these two cases, `Align_DMA_Buffer` first attempts to align the buffer so that the data completely avoids a page boundary. If **bufferSize** is not big enough for complete alignment, the function attempts to partially align the data to ensure that a page boundary does not cause unpredictable results. Partial alignment is possible if **bufferSize** ≥ **count** + 1. If neither form of alignment is possible, the function returns an error. If `Align_DMA_Buffer` partially aligned the data, the function returns a **memPageError** warning indicating that a page boundary is still in the data.

☞ **Note:** ***Physical DMA page boundaries do not exist on EISA bus computers. However, page boundaries may be introduced on these computers as a side effect of Windows 386 Enhanced mode and the Windows NT virtual memory management system. This happens when a buffer is locked into physical memory in preparation for a DAQ operation. If the memory manager cannot find a contiguous space large enough, it will fragment the buffer, placing pieces of it here and there in physical memory. This type of page boundary will only affect the performance on an AT bus computer. NI-DAQ will use the DMA chaining feature available on EISA computers to*** *CHAIN ACROSS* ***page boundaries, thus avoiding the delay involved in DMA programming.***

You should call `Align_DMA_Buffer` *after* NI-DAQ has loaded **buffer** with the data samples (for waveform generation or digital output) and *before* calling `WFM_Op`, `WFM_Load`, `DIG_Block_In`, or `DIG_Block_Out`. You should pass the aligned buffer to the waveform generation and pattern generation functions the *same* way you would an unaligned buffer. The **count** parameter in the waveform generation or pattern

Align_DMA_Buffer

Continued

generation function call should be the same as the **count** parameter passed to `Align_DMA_Buffer`, not **bufferSize**.

If you want to access the data in **buffer** after calling `Align_DMA_Buffer`, access the data starting at **buffer**[**alignIndex**] (if your array is zero based).

After using an aligned buffer for waveform generation or pattern generation, NI-DAQ *unaligns* the data. After the buffer has been unaligned, the first data sample is at offset zero of the buffer again. If you want to use the buffer for waveform generation or pattern generation again after it has been unaligned, you must make another call to `Align_DMA_Buffer` before calling `WFM_Op`, `WFM_Load`, `DIG_Block_In`, or `DIG_Block_Out`.

See *Waveform Generation Application Hints* and *Digital I/O Application Hints* in Chapter 3*, Software Overview*, of the *NI-DAQ User Manual for PC Compatibles* for more information on the use of `Align_DMA_Buffer`.

See Chapter 4*, DMA and Programmed I/O Performance Limitations*, of the *NI-DAQ User Manual for PC Compatibles* for a discussion of DMA page boundaries and special run-time considerations.

AO_Calibrate

Format

status = AO_Calibrate (deviceNumber, operation, EEPROMloc)

Purpose

Loads a set of calibration constants into the calibration DACs or copies a set of calibration constants from one of four EEPROM areas to EEPROM area 1. You can load an existing set of calibration constants into the calibration DACs from a storage area in the onboard EEPROM. You can copy EEPROM storage areas 2 through 5 (EEPROM area 5 contains the factory calibration constants) to storage area 1. NI-DAQ automatically loads the calibration constants stored in EEPROM area 1 the first time a function pertaining to the AT-AO-6/10 is called.

☞ **Note:** ***Use the calibration utility provided with the AT-AO-6/10 to perform a calibration procedure. Refer to the calibration chapter in the*** *AT-AO-6/10 User Manual* ***for more information regarding the calibration procedure.***

Parameters

Input

Name	Type		Description
	Windows	Windows NT	
deviceNumber	I16	I32	assigned by configuration utility
operation	I16	I32	operation to be performed
EEPROMloc	I16	I32	storage location in the onboard EEPROM

Parameter Discussion

operation determines the operation to be performed.

1: Load calibration constants from **EEPROMloc**.

2: Copy calibration constants from **EEPROMloc** to EEPROM user calibration area 1.

AO_Calibrate

Continued

EEPROMloc selects the storage location in the onboard EEPROM to be used. You can use different sets of calibration constants to compensate for configuration or environmental changes.

1:	User calibration area 1.
2:	User calibration area 2.
3:	User calibration area 3.
4:	User calibration area 4.
5:	Factory calibration area.

Using This Function

When NI-DAQ initializes the AT-AO-6/10, the DAC calibration constants stored in **EEPROMloc** 1 (user calibration area 1) provide the gain and offset values used to ensure proper device operation. In other words, `Init_DA_Brds` performs the equivalent of calling `AO_Calibrate` with operation set to 1 and **EEPROMloc** set to 1. When the AT-AO-6/10 leaves the factory, **EEPROMloc** 1 contains a copy of the calibration constants stored in **EEPROMloc** 5, the factory area.

A calibration procedure performed in bipolar mode is not valid for unipolar and vice versa. Please see the calibration chapter of the *AT-AO-6/10 User Manual* for more information regarding calibrating the device.

AO_Change_Parameter

Format

status = AO_Change_Parameter (deviceNumber, channel, paramID, paramValue)

Purpose

Selects a specific parameter setting for the analog output section of the device or an analog output channel (AT-MIO-16E-2, AT-MIO-64E-3, NEC-MIO-16E-4, AO-2DC devices, and AT-MIO-16X only). You can select parameters related to analog output not listed here through the `AO_Configure` function.

☞ **Note:** ***NI-DAQ for Windows NT does not support this function.***

Parameters

Input

Name	Type		Description
	Windows	Windows NT	
deviceNumber	I16	n/a	assigned by configuration utility
channel	I16	n/a	number of channel you want to configure; you can use -1 to indicate all channels
paramID	U32	n/a	identification of the parameter you want to change
paramValue	U32	n/a	new value for the parameter specified by **paramID**

Parameter Discussion

Legal ranges for **paramID** and **paramValue** are given in terms of constants defined in a header file. The header file you should use depends on the language you are using:

- C programmers—`NIDAQCNS.H` (`DATAACQ.H` for LabWindows/CVI)

AO_Change_Parameter

Continued

- BASIC programmers—`NIDAQCNS.INC` (Visual Basic for Windows programmers should refer to the *Programming Language Considerations* section in Chapter 1 for more information.)
- Pascal programmers—`NIDAQCNS.PAS`

Legal values for **channel** depend on the type of device you are using; analog output channels are labeled 0 through *n*-1, where *n* is the number of analog output channels on your device. You can set **channel** to -1 to indicate that you want the same parameter selection for all channels. You must set **channel** to -1 if you want to change a parameter you cannot change on per-channel basis.

Legal values for **paramValue** depend on **paramID**. The following paragraphs list features you can configure along with legal values for **paramID** with explanations and corresponding legal values for **paramValue**.

Reglitching

Every time you change the state of your DAC, a very small glitch is generated in the signal generated by the DAC. When reglitching is turned off, glitch size depends on the binary patterns that are written into the DAC; the glitch is largest when the most significant bit in the pattern changes (when the waveform crosses the midrange of the DAC); it is smaller in other cases. When reglitching is turned on, the glitch size is much less dependent on the bit pattern.

To change the reglitching parameter, set **paramID** to `ND_REGLITCH`.

If you are not concerned about this, you are likely to be satisfied by the default values NI-DAQ selects for you if you do not call this function. The following table lists devices on which you can change this parameter

AO_Change_Parameter

Continued

Device Type	Per-Channel Selection Possible	Legal Range for paramValue	Default Setting for paramValue
AT-MIO-16X[1]	No	`ND_OFF` and `ND_ON`	`ND_ON`
AT-MIO-16E-1 AT-MIO-16E-2 AT-MIO-64E-3 NEC-MIO-16E-4	Yes	`ND_OFF` and `ND_ON`	`ND_OFF`
[1]***Warning:** **If you turn off reglitching on the AT-MIO-16X, timing problems that NI-DAQ cannot detect may occur.***			

Voltage or Current Output

Some devices require separate calibration constants for voltage and current outputs. Setting the output type to voltage or current for these devices causes the driver to use the correct calibration constants and to interpret the input data correctly in `AO_VWrite`. To change the output type, set **paramID** to `ND_OUTPUT_TYPE`.

Device Type	Per-Channel Selection Possible	Legal Range for paramValue	Default Setting for paramValue
PC-AO-2DC DAQCard-AO-2DC	Yes	`ND_CURRENT` and `ND_VOLTAGE`	`ND_VOLTAGE`

Using This Function

This function lets you customize the behavior of the analog output section of your device. You should call this function before calling NI-DAQ functions that cause output on the analog output channels. You can call this function as often as you need.

AO_Configure

Format

status = AO_Configure (deviceNumber, chan, outputPolarity, intOrExtRef, refVoltage, updateMode)

Purpose

Informs NI-DAQ of the output range and polarity selected for each analog output channel on the device and indicates the update mode of the DACs. Use this function if you have changed the jumper settings affecting analog output range and polarity from their factory settings. For the MIO E Series devices, AT-MIO-64F-5, AT-MIO-16X, SCXI-1200, DAQPad-1200, DAQCard-1200, Lab-PC-1200, and AO-2DC devices, which have no jumpers for analog output configuration, this function programs the device for the settings you want.

If you are using NI-DAQ for Windows and you have recorded a nondefault analog output configuration through the NI-DAQ Configuration Utility, you do not need to use `AO_Configure` because NI-DAQ uses the settings recorded by the NI-DAQ Configuration Utility. If you have a software-configurable device, you can use `AO_Configure` to change the analog output configuration on the fly.

Warning: ***For the AT-AO-6/10, NI-DAQ records the configuration information for output polarity and update mode in channel pairs. A call to*** `AO_Configure` ***will record the same output polarity and update mode selections for both channels in a pair.***

AO_Configure

Continued

Parameters

Input

Name	Type		Description
	Windows	Windows NT	
deviceNumber	I16	I32	assigned by configuration utility
chan	I16	I32	analog output channel number
outputPolarity	I16	I32	unipolar or bipolar
intOrExtRef	I16	I32	reference source
refVoltage	F64	F64	voltage reference value
updateMode	I16	I32	when to update the DACs

Parameter Discussion

chan is the analog output channel number.

Range:	0 or 1 for the AO-2DC, Lab and 1200 Series analog output devices, and MIO devices.
	0 through 5 for the AT-AO-6.
	0 through 9 for the AT-AO-10.

outputPolarity indicates whether the analog output channel is configured for unipolar or bipolar operation.

For the AT-AO-6/10 and MIO devices (except the MIO-16XE-50 devices):

0:	Bipolar operation (default setting, output range is from **-refVoltage** to **+refVoltage**).
1:	Unipolar operation (output range is from 0 to **+refVoltage**).

For the Lab and 1200 Series analog output devices:

0:	Bipolar operation (default setting, output range is from -5 to +5 V).
1:	Unipolar operation (output range is from 0 to +10 V).

AO_Configure

Continued

For the MIO-16XE-50 devices:

- 0: Bipolar operation (output range is from -10 to +10 V).

For the AO-2DC devices:

- 0: Bipolar operation (output range is from -5 to +5 V).
- 1: Unipolar operation (default setting, output range is from 0 to +10 V or 0 to 20 mA).

intOrExtRef indicates the source of voltage reference.

- 0: Internal reference.
- 1: External reference.

The MIO devices, except the 16-bit MIO E Series devices, and AT-AO-6/10 devices support external analog output voltage references.

refVoltage is the analog output channel voltage reference value. You can configure each channel to use an internal reference of +10 V (the default) or an external reference. Although each pair of channels is served by a single external reference connection, the configuration of the external reference operates on a per-channel basis. Therefore it is possible to have one channel in a pair internally referenced and the other channel in the same pair externally referenced.
Range: -10 to +10 V.

If you make a reference voltage connection, you must assign **refVoltage** the value of the external reference voltage in a call to `AO_Configure` for the `AO_VWrite` and `AO_VScale` functions to operate properly. For devices that have no external reference pin, the output range is determined by **outputPolarity**, and NI-DAQ ignores this parameter.

updateMode indicates whether an analog output channel is updated when written to:

- 0: Updated when written to (default setting).
- 1: Not updated when written to, but updated later after a call to `AO_Update` (later internal update mode).
- 2: Not updated when written to, but updated upon application of an active low pulse to:
  - The OUT2 pin for the MIO-16 and MIO-16D
  - The EXTDACUPDATE pin for the AT-MIO-16F-5, AT-MIO-64F-5, and AT-MIO-16X

AO_Configure

Continued

- The EXTUPDATE pin for the AT-AO-6/10 and Lab and 1200 Series analog output devices (later external update mode)
- The PFI5 pin for the MIO E Series devices. To alter the pin and polarity selections for an MIO E Series device, you can call `Select_Signal` with **signal** = `ND_OUT_UPDATE` after you call `AO_Configure`.

Using This Function

`AO_Configure` stores information about the analog output channel on the specified device in the configuration table for the analog channel. For the AT-AO-6/10, the **outputPolarity** and **updateMode** information is stored for channel pairs. For example, analog output channels 0 and 1 are grouped in a channel pair, and a call to `AO_Configure` for channel 0 will record the **outputPolarity** and **updateMode** for both channels 0 and 1. Likewise, a call to `AO_Configure` for channel 1 will record the **outputPolarity** and **updateMode** for both channels 0 and 1. The AT-AO-6/10 channel pairs are as follows:

AT-AO-6/10 channel pairs:

- Channels 0 and 1.
- Channels 2 and 3.
- Channels 4 and 5.
- Channels 6 and 7 (AT-AO-10 only).
- Channels 8 and 9 (AT-AO-10 only).

The analog output channel configuration tables default to the following:

- MIO device and AT-AO-6/10:
  outputPolarity = 0: Bipolar.
  refVoltage = 10 V.
  updateMode = 0: Update when written to.
- Lab and 1200 Series analog output devices:
  outputPolarity = 0: Bipolar (-5 to +5 V).
  updateMode = 0: Updated when written to.

If you configure an output channel for later internal update mode (**updateMode** = 1), you can configure no other output channels for later external update mode (**updateMode** = 2). Likewise, if you configure an output channel for later external update mode, you can configure no other output channels for later internal update mode.

AO_Configure

Continued

If the physical configuration (the jumpered settings) of the analog output channels on your device differs from the default setting, you must call `AO_Configure` with the true configuration information for the remaining analog output functions to operate properly.

☞ **Note:** ***The AT-AO-6/10 allows you to physically configure each analog output channel (the jumper setting) for bipolar or unipolar operation. To ensure proper operation, you should configure both channels in a channel pair the same way.***

On the AT-MIO-16X, AT-MIO-64F-5, and MIO E Series devices (except MIO-16XE-50 devices), the calibration constants used for analog output change depending on the polarity of the analog output channels. NI-DAQ always ensures that the calibration constants in use match the current polarity of the channels.

If you change the polarity on the AT-MIO-16X or the AT-MIO-64F-5 by calling `AO_Configure`, NI-DAQ checks if the load area contains the appropriate constants. If so, NI-DAQ loads the constants from the load area. Otherwise, NI-DAQ will load the constants from the factory area for the current polarity and return status code **calConstPolarityConflictError**. The actual loading of calibration constants will take place when you call an AO or WFM function. See the `Calibrate_E_Series` function description for information about calibration constant loading on the MIO E Series devices.

If you want to load constants from some other EEPROM area, you must call the `MIO_Calibrate` function after calling `AO_Configure`.

AO_Update

Format

status = AO_Update (deviceNumber)

Purpose

Updates analog output channels on the specified device to new voltage values when the later internal update mode is enabled by a previous call to `AO_Configure`.

Parameters

Input

Name	**Type**		**Description**
	Windows	**Windows NT**	
deviceNumber	I16	I32	slot or device ID number

Using This Function

`AO_Update` issues an update pulse to all analog output channels on the specified device. All analog output channel voltages then simultaneously change to the last value written.

AO_VScale

Format

status = AO_VScale (deviceNumber, chan, voltage, binVal)

Purpose

Scales a voltage to a binary value that, when written to one of the analog output channels, produces the specified voltage.

Parameters

Input

Name	Type		Description
	Windows	Windows NT	
deviceNumber	I16	I32	assigned by configuration utility
chan	I16	I32	analog output channel number
voltage	F64	F64	voltage, in volts, to be converted to a binary value

Output

Name	Type		Description
	Windows	Windows NT	
binVal	I16	I16	converted binary value returned

Parameter Discussion

chan is the analog output channel number.

Range:	0 or 1 for the AT-DSP2200, Lab and 1200 Series analog output devices, and MIO devices.
	0 through 5 for AT-AO-6.

AO_VScale

Continued

0 through 10 for AT-AO-10.

☞ **Note:** ***C Programmers*—binVal *is a pass-by-reference parameter.***

Using This Function

Using the following formula, `AO_VScale` calculates the binary value to be written to the specified analog output channel to generate an output voltage corresponding to **voltage**.

binVal = (**voltage**/**refVoltage**) $*$ maxBinVal

where values of **refVoltage** and maxBinVal are listed in the following table:

Device	Unipolar		Bipolar	
	refVoltage	maxBinVal	**refVoltage**	maxBinVal
Most MIO devices	*	4,096	*	2,048
AT-MIO-16X, AT-MIO-16XE-50	*	65,536	*	32,768
Lab and 1200 Series analog output devices	10.0	4,096	5.0	2,048
AT-AO-6/10	*	4,096	*	2,048
AT-DSP2200	—	—	2.828	32,768
Note: **** indicates that you specify the value of refVoltage in the*** `AO_Configure` ***function call.***				

Notice that **refVoltage** is the value you specify in `AO_Configure` (default is 10 V for the AT-MIO-16 and AT-AO-6/10). Because you can independently configure the analog output channels for range and polarity, NI-DAQ can translate the same voltage to different values for each channel.

AO_VWrite

Format

status = AO_VWrite (deviceNumber, chan, voltage)

Purpose

Accepts a floating-point voltage value, scales it to the proper binary number, and writes that number to an analog output channel to change the output voltage.

Parameters

Input

Name	Type		Description
	Windows	Windows NT	
deviceNumber	I16	I32	assigned by configuration utility
chan	I16	I32	analog output channel number
voltage	F64	F64	floating-point value to be scaled and written

Parameter Discussion

chan is the analog output channel number.

Range: 0 or 1 for the AT-DSP2200 and AO-2DC, Lab and 1200 Series analog output, and MIO devices.
0 through 5 for AT-AO-6.
0 through 9 for AT-AO-10.

voltage is the floating-point value to be scaled and written to the analog output channel. The range of voltages depends on the type of device, on the jumpered output polarity and, for the MIO device and AT-AO-6/10, on whether you apply an external voltage reference.

AO_VWrite

Continued

- Default ranges (bipolar, internal voltage reference):

– MIO device:	-10 to +10 V.
– AT-AO-6/10:	-10 to +10 V.
– Lab and 1200 Series analog output devices:	-5 to +5 V.
– AT-DSP2200:	-2.828 to +2.828 V.

- Default ranges (unipolar, internal voltage reference):

– AO-2DC device:	0 to +10 V.

If you set the output type to current by calling `AO_Change_Parameter`, the floating-point value indicates the current in amps.

- Default ranges (unipolar, internal voltage reference):

– AO-2DC device:	0 to 0.002 A.

Using This Function

`AO_VWrite` scales **voltage** to a binary value and then writes that value to the DAC in the analog output channel. If the analog output channel is configured for immediate update, the output voltage or current changes immediately. Otherwise, the output voltage or current changes when NI-DAQ issues an update command or pulse.

If you have changed the output polarity for the analog output channel from the factory setting of bipolar to unipolar, you must call `AO_Configure` with this information for `AO_VWrite` to correctly scale the floating-point value to the binary value.

AO_Write

Format

status = AO_Write (deviceNumber, chan, value)

Purpose

Writes a binary value to one of the analog output channels, changing the voltage produced at the channel.

Parameters

Input

Name	Type		Description
	Windows	Windows NT	
deviceNumber	I16	I32	assigned by configuration utility
chan	I16	I32	analog output channel number
value	I16	I16	digital value to be written

Parameter Discussion

chan is the analog output channel number.

Range:	0 or 1 for Lab and 1200 Series analog output and MIO devices and the AT-DSP2200.
	0 through 5 for AT-AO-6.
	0 through 9 for AT-AO-10.

value is the digital value to be written to the analog output channel. **value** has several ranges, depending on whether the analog output channel is configured for unipolar or

AO_Write

Continued

bipolar operations and on the analog output resolution of the device as shown in the following table:

Device	Bipolar	Unipolar
Most devices	-2,048 to +2,047	0 to +4,095
AT-MIO-16X, 16-bit MIO E Series devices, AT-DSP2200	-32,768 to +32,767	0 to +65,536

Using This Function

`AO_Write` writes **value** to the DAC in the analog output channel. If you configure the analog output channel for immediate update, the output voltage or current changes immediately. Otherwise, the output voltage or current changes when NI-DAQ issues an update command or pulse.

Calibrate_1200

Format

status = Calibrate_1200 (device, calOP, saveNewCal, EEPROMloc, calRefChan, grndRefChan, DAC0chan, DAC1chan, calRefVolts, gain)

Purpose

The DAQCard-1200, DAQPad-1200, Lab-PC-1200, Lab-PC-1200AI, and SCXI-1200 come fully equipped with accurate factory calibration constants. However, if you feel that the device is not performing either analog input or output accurately and suspect the device calibration to be in error, you can use `Calibrate_1200` to obtain a user defined set of new calibration constants.

A complete set of calibration constants consists of ADC constants for all gains at one polarity plus DAC constants for both DACs, again at the same polarity setting. It is important to understand the polarity rules. The polarity your device was in when a set of calibration constants was created must match the polarity your device is in when those calibration constants are used. For example, calibration constants created when your ADC is in unipolar must only be used for data acquisition when your ADC is also in unipolar.

You can store up to six sets of user-defined calibration constants. These are stored in the EEPROM on your device in places called user calibration areas. Refer to your hardware user manual for more information on these calibration areas. You may also at any time use the calibration constants created at the factory. These are stored in write protected places in the EEPROM called factory calibration areas. There are two of these. One holds constants for bipolar operation and the other for unipolar. One additional area in the EEPROM important to calibration is called the default load table. This table contains four pointers to sets of calibration constants; one pointer each for ADC unipolar constants, ADC bipolar constants, DAC unipolar and DAC bipolar. This table is used by NI-DAQ for calibration constant loading.

It is also important to understand the calibration constant loading rules. The first time a function requiring use of the ADC or DAC is called in an application, NI-DAQ automatically loads a set of calibration constants. At that time, the polarities of your ADC and DACs are examined and the appropriate pointers in the default load table are used. The calibration constant loading will be done once after the DLL is loaded. If your DLL is ever unloaded and then reloaded again, the calibration constant loading will also be done again.

☞ **Note:** ***NI-DAQ for Windows NT does not support this function.***

Calibrate_1200

Continued

Note: ***Calling this function on an SCXI-1200 with Remote SCXI may take an extremely long time. We strongly recommend that you switch your SCXI-1200 to use a parallel port connection before performing the calibration and store the calibration constants in one of the EEPROM storage locations.***

Warning: ***Read the calibration chapter in your device user manual before using*** `Calibrate_1200`.

Parameters

Input

Name	Type		Description
	Windows	Windows NT	
device	I16	n/a	device number
calOP	I16	n/a	operation to be performed
saveNewCal	I16	n/a	save new calibration constants
EEPROMloc	I16	n/a	storage location on EEPROM
calRefChan	I16	n/a	AI channel connected to the calibration voltage
grndRefChan	I16	n/a	AI channel that is grounded
DAC0chan	I16	n/a	AI channel connected to DAC0
DAC1chan	I16	n/a	AI channel connected to DAC1
calRefVolts	F64	n/a	DC calibration voltage
gain	F64	n/a	gain at which ADC is operating

Calibrate_1200

Continued

Parameter Discussion

calOP determines the operation to be performed.

1: Load calibration constants from **EEPROMloc**. If **EEPROMloc** is 0, the default load table is used and NI-DAQ will ensure that the constants loaded will be appropriate for the current polarity settings. If **EEPROMloc** is any other value you must ensure that the polarity of your device matches those of the calibration constants.

2: Calibrate the ADC using DC reference voltage **calRefVolts** connected to **calRefChan**. To calibrate the ADC, you must ground one input channel (**grndRefChan**) and connect a voltage reference between any other channel and AGND (pin 11). *Please remember that the ADC must be in referenced single-ended mode for successful calibration of the ADC.* After calibration, the calibration constants that were obtained during the process will remain in use by the ADC until the device is initialized again.

3: Calibrate the DACs. **DAC0chan** and **DAC1chan** are the analog input channels to which DAC0 and DAC1 are connected, respectively. To calibrate the DACs, you must wrap-back the DAC0 out (pin 10) and DAC1 out (pin 12) to any two analog input channels. *Please remember that the ADC must be in referenced single-ended and bipolar mode and fully calibrated (using* ***calOP****=2) for successful calibration of the DACs.* After calibration, the calibration constants that were obtained during the process will remain in use by the DACs until the device is initialized again.

4: Reserved.

5: Edit the default load table so that the set of constants in the area identified by **EEPROMloc** (1–6, 9 or 10) become the default calibration constants for the ADC. NI-DAQ will change either the unipolar or bipolar pointer in the default load table depending on the polarity those constants are intended for. The factory default for the ADC unipolar pointer is **EEPROMloc**=9. The factory default for the ADC bipolar pointer is **EEPROMloc**=10. You can specify any user area in **EEPROMloc** after you have run a calibration on the ADC and saved the calibration constants to that user area. Or you can specify **EEPROMloc**=9 or 10 to reset the default load table to the factory calibration for unipolar and bipolar mode respectively.

6: Edit the default load table so that the set of constants in the area identified by **EEPROMloc** (1–6, 9 or 10) become the default calibration constants for the DACs. NI-DAQ's behavior for **calOP**=6 is identical to that for **calOP**=5. Just substitute DAC everywhere you see ADC.

Calibrate_1200

Continued

The following table summarizes the possible values of other parameters depending on the value of **calOP**:

Table 2-2. Possible Calibrate_1200 Parameter Values

calOP	saveNewCal	EEPROMloc	calRefChan	grndRefChan	DAC0chan	DAC1chan	calRefVolts	gain
1	ignored	0–10	ignored	ignored	ignored	ignored	ignored	ignored
2	0 or 1	1–6	AI chan connected to voltage source (0–7)	AI chan connected to ground (0–7)	ignored	ignored	the voltage of the voltage source	1, 2, 5, 10, 50, or 100
3	0 or 1	1–6	ignored	ignored	AI chan connected to DAC0Out (0–7)	AI chan connected to DAC1Out (0–7)	ignored	1, 2, 5, 10, 50, or 100
5	ignored	1–6, 9–10	ignored	ignored	ignored	ignored	ignored	ignored
6	ignored	1–6, 9–10	ignored	ignored	ignored	ignored	ignored	ignored

saveNewCal is only valid when **calOP** is 2 or 3.

0: Do not save new calibration constants. Even though not permanently saved in the EEPROM, calibration constants created after a successful calibration will remain in use by your device until your device is initialized again.

1: Save new calibration constants in **EEPROMloc** (1–6).

EEPROMloc selects the storage location in the onboard EEPROM to be used. Different sets of calibration constants can be used to compensate for configuration or environmental changes.

0: Use the default load table (only valid if **calOP** = 1).

1: User calibration area 1.

2: User calibration area 2.

3: User calibration area 3.

Calibrate_1200

Continued

4:	User calibration area 4.
5:	User calibration area 5.
6:	User calibration area 6.
7:	Invalid.
8:	Invalid.
9:	Factory calibration area for unipolar (write protected).
10:	Factory calibration area for bipolar (write protected).

Notice that the user cannot write into **EEPROMloc** 9 and 10.

calRefChan is the analog input channel connected to the calibration voltage of **calRefVolts** when **calOP** is 2.
Range: 0 through 7.

grndRefChan is the analog input channel connected to ground when **calOP** is 2.
Range: 0 through 7.

DAC0chan is the analog input channel connected to DAC0 when **calOP** is 3.
Range: 0 through 7.

DAC1chan is the analog input channel connected to DAC1 when **calOP** is 3.
Range: 0 through 7.

calRefVolts is the value of the DC calibration voltage connected to **calRefChan** when **calOP** = 2. *If you are calibrating at a gain other than 1, make sure you apply a voltage so that calRefVolts * gain is within the upper limits of the analog input range of the device.*

gain is the device gain setting at which you want to calibrate when **calOP** is 2 or 3. When you perform an analog input operation, a calibration constant for that gain must be available. *When you run the* `Calibrate_1200` *function at a particular gain, the device can only be used to collect data accurately at that gain.* If you are creating a set of calibration constants that you intend to use then you must be sure to calibrate at all gains at which you intend to sample.
Range: 1, 2, 5, 10, 50, or 100.

Using This Function

A calibration performed in bipolar mode is not valid for unipolar and vice versa. `Calibrate_1200` performs a bipolar or unipolar calibration, or loads the bipolar or unipolar constants (**calOP**=1, **EEPROMloc**=0), depending on the value of the polarity

Calibrate_1200

Continued

parameter in the last call to `AI_Configure` and `AO_Configure`. If analog input measurements are taken with the wrong set of calibration constants loaded, you may get erroneous data.

Calibrate for a particular gain if you plan to acquire at that gain. If you calibrate the device yourself make sure you calibrate at a gain that you are likely to use. Each gain has a different calibration constant. When you switch gains, NI-DAQ will automatically load the calibration constant for that particular gain. If you have not calibrated for that gain and saved the constant earlier, an incorrect value will be used.

To set up your own calibration constants in the user area for both unipolar and bipolar configurations:

The basic steps are to create and store both unipolar and bipolar ADC calibration constants, and modify the default load table so that NI-DAQ will automatically load your constants instead of the factory constants.

Step 1. Unipolar calibration—Change the polarity of your device to unipolar (by using the `AI_Configure` call or use the NI-DAQ Configuration Utility in Windows). Call `Calibrate_1200` to perform an ADC calibration, as in the following example:

```
status = Calibrate_1200 (device, 2, 1, EEPROMloc, calRefChan,
grndRefChan, 0, 0, calRefVolts, gain)
```

where you specify **device**, **EEPROMloc** (say 1, for example), **calRefChan**, **grndRefChan**, **calRefVolts**, and **gain**.

Next call this function again; for example:

```
status = Calibrate_1200 (device, 5, 0, EEPROMloc, 0, 0, 0, 0, 0, 0)
```

where the **device** and **EEPROMloc** are the same as in the first function call.

NI-DAQ will automatically modify the ADC unipolar pointer in the default load table to point to user area 1.

Step 2. Bipolar calibration—Change the polarity of your device to bipolar. Call `Calibrate_1200` to perform another ADC calibration (**calOP**=2) with **saveNewCal**=1 (save) and **EEPROMloc** set to a different user area (say, 2) as shown above. Next call the function with **calOP**=5 and **EEPROMloc**=2 as shown above. NI-DAQ will automatically modify the ADC bipolar pointer in the default load table to point to user area 2. At this point, you have set up user area 1 to be your default load

Calibrate_1200

Continued

area when you operate the device in unipolar mode and user area 2 to be your default load area when you operate the device in bipolar mode. The loading of the appropriate constants will be handled automatically by NI-DAQ.

Failed calibrations leave your device in an incorrectly calibrated state. If you run this function with **calOp**=2 or 3 and receive an error, you must reload a valid set of calibration constants. If you have a valid set of user defined constants in one of the user areas you can load them. Otherwise you should reload the factory constants.

Calibrate_E_Series

Format

status = Calibrate_E_Series (deviceNumber, calOP, setOfCalConst, calRefVolts)

Purpose

Use this function to calibrate your E Series device and to select a set of calibration constants to be used by NI-DAQ.

Warning: ***Read the calibration chapter in your device user manual before using*** `Calibrate_E_Series`**.**

Note: ***Analog output channels and the*** `AO` ***and*** `WFM` ***functions do not apply to the AI E Series devices.***

Parameters

Input

Name	Type		Description
	Windows	Windows NT	
deviceNumber	I16	n/a	assigned by configuration utility
calOP	U32	n/a	operation to be performed
setOfCalConst	U32	n/a	set of calibration constants or the EEPROM location to use
calRefVolts	F64	N/A	DC calibration voltage

Parameter Discussion

The legal ranges for the **calOp** and **setOfCalConst** parameters are given in terms of constants that are defined in a header file. The header file you should use depends on the language you are using:

- C programmers—`NIDAQCNS.H` (`DATAACQ.H` for LabWindows/CVI)
- BASIC programmers—`NIDAQCNS.INC`

Calibrate_E_Series

Continued

- Pascal programmers—`NIDAQCNS.PAS`

calOP determines the operation to be performed.
Range:

`ND_SET_DEFAULT_LOAD_AREA`:	Make **setOfCalConst** the default load area; do not perform calibration.
`ND_SELF_CALIBRATE`:	Perform self-calibration of the device.
`ND_EXTERNAL_CALIBRATE`:	Perform external calibration of the device.

setOfCalConst selects the set of calibration constants to be used by NI-DAQ. These calibration constants reside in the onboard EEPROM or are maintained by NI-DAQ.
Range:

`ND_FACTORY_EEPROM_AREA`:	Factory calibration area of the EEPROM. You cannot modify this area, so you can set **setOfCalConst** to `ND_FACTORY_EEPROM_AREA` only when **calOP** is set to `ND_SET_DEFAULT_LOAD_AREA`.
`ND_NI_DAQ_SW_AREA`:	NI-DAQ maintains calibration constants internally; no writing into the EEPROM occurs. You cannot use this setting when **calOP** is set to `ND_SET_DEFAULT_LOAD_AREA`. This setting is useful if you want to calibrate your device repeatedly during your program, and you do not want to store the calibration constants in the EEPROM.
`ND_USER_EEPROM_AREA`:	For the user calibration area of the EEPROM. If **calOP** is set to `ND_SELF_CALIBRATE` or `ND_EXTERNAL_CALIBRATE`, the new calibration constants will be written into this area, and this area will become the new default load area. You can use this setting if you want to run several NI-DAQ applications during one measurement session conducted at same temperature, and you do not want to recalibrate your device in each application.

calRefVolts is the value of the DC calibration voltage connected to analog input channel 0 when **calOP** is `ND_EXTERNAL_CALIBRATE`. This parameter is ignored when **calOP** is `ND_SET_DEFAULT_LOAD_AREA` or `ND_SELF_CALIBRATE`.

Calibrate_E_Series

Continued

Range:
12-bit E Series devices: +6.0 to +10.0 V
16-bit E Series devices: +6.0 to +9.999 V

Using This Function

Your device contains calibration D/A converters (calDACs) that are used for fine-tuning the analog circuitry. The calDACs must be programmed (loaded) with certain numbers called calibration constants. Those constants are stored in non-volatile memory (EEPROM) on your device or are maintained by NI-DAQ. To achieve specification accuracy, you should perform an internal calibration of your device just before a measurement session but after your computer and the device have been powered on and allowed to warm up for at least 15 minutes. Frequent calibration produces the most stable and repeatable measurement performance. The device is not harmed in any way if you recalibrate it as often as you like.

Two sets of calibration constants can reside in two *load areas* inside the EEPROM; one set is programmed at the factory, and the other is left for the user. One load area in the EEPROM corresponds to one set of constants. The load area NI-DAQ uses for loading calDACs with calibration constants is called the default load area. When you get the device from the factory, the default load area is the area that contains the calibration constants obtained by calibrating the device in the factory. NI-DAQ automatically loads the relevant calibration constants stored in the load area the first time you call a function (an `AI`, `AO`, `DAQ`, `SCAN` and `WFM` function) that requires them. NI-DAQ also automatically reloads calibration constants whenever appropriate; see the *Calibration Constant Loading by NI-DAQ* section later in this function for details. When you call the `Calibrate_E_Series` function with **setOfCalConst** set to `ND_NI_DAQ_SW_AREA`, NI-DAQ uses a set of constants it maintains in a load area that does not reside inside the EEPROM.

Note: ***Calibration of your MIO or AI device takes some time. Do not be alarmed if the*** `Calibrate_E_Series` ***function takes several seconds to execute.***

Note: ***After powering on your computer, you should wait for some time (typically 15 minutes) for the entire system to warm up before performing the calibration. You should allow the same warm-up time before any measurement session that will take advantage of the calibration constants determined by using the*** `Calibrate_E_Series` ***function.***

Calibrate_E_Series

Continued

Warning: ***When you call the*** `Calibrate_E_Series` ***function with* calOP *set to*** `ND_SELF_CALIBRATE` ***or*** `ND_EXTERNAL_CALIBRATE`***, NI-DAQ will abort any ongoing operations the device is performing and set all configurations to defaults. Therefore we recommend that you call*** `Calibrate_E_Series` ***before calling other NI-DAQ functions (except*** `USE` ***functions) or when no other operations are going on.***

BASIC programmers should set aside at least 18,000 bytes for this function to perform calibration. This is done with the `SETMEM` statement, for example:

```
Heap.Size = SETMEM(-18000)
```

Explanations about using this function for different purposes (with different values of **calOP)** are given in the following sections.

Changing the Default Load Area

Set **calOP** to `ND_SET_DEFAULT_LOAD_AREA` if you want to change the area used for calibration constant loading. The storage location selected by **setOfCalConst** becomes the new default load area.

Example:

You want to make the factory area of the EEPROM default load area. You should make the following call:

```
Calibrate_E_Series(deviceNumber, ND_SET_DEFAULT_LOAD_AREA,
ND_FACTORY_EEPROM_AREA, 0.0)
```

Performing Self-Calibration of the Board

Set **calOP** to `ND_SELF_CALIBRATE` if you want to perform self-calibration of your device. The storage location selected by **setOfCalConst** becomes the new default load area.

Calibrate_E_Series

Continued

Example:

You want to perform self-calibration of your device and you want to store the new set of calibration constants in the user area of the EEPROM. You should make the following call:

```
Calibrate_E_Series(deviceNumber, ND_SELF_CALIBRATE,
ND_USER_EEPROM_AREA, 0.0)
```

The EEPROM user area will become the default load area.

Performing External Calibration of the Board

Set **calOP** to `ND_EXTERNAL_CALIBRATE` if you want to perform external calibration of your device. The storage location selected by **setOfCalConst** becomes the new default load area.

Make the following connections before calling the `Calibrate_E_Series` function:

12-bit MIO E Series Devices	16-bit MIO E Series and All AI E Series
1. Connect the positive output of your reference voltage source to the analog input channel 8. 2. Connect the negative output of your reference voltage source to the AISENSE line. 3. Connect the DAC0 line (analog output channel 0) to analog input channel 0. 4. If your reference voltage source and your computer are floating with respect to each other, connect the AISENSE line to the AIGND line as well as to the negative output of your reference voltage source.	1. Connect the positive output of your reference voltage source to analog input channel 0. 2. Connect the negative output of your reference voltage source to analog input channel 8. ***Note:*** ***By performing these first two connections, you supply the reference voltage to analog input channel 0, which is configured for differential operation.*** 3. If your reference voltage source and your computer are floating with respect to each other, connect the negative output of your reference voltage source to the AIGND line as well as to analog input channel 8.

Calibrate_E_Series

Continued

Example:

You want to perform an external calibration of your device using an external reference voltage source with a precise 7.0500 V reference, and you want NI-DAQ to maintain a new set of calibration constants without storing them in the EEPROM. You should make the following call:

```
Calibrate_E_Series(deviceNumber, ND_EXTERNAL_CALIBRATE,
ND_NI_DAQ_SW_AREA, 7.0500)
```

The internal NI-DAQ area will become the default load area, and the calibration constants will be lost when your application ends.

Calibration Constant Loading by NI-DAQ

NI-DAQ automatically loads calibration constants into calDACs whenever you call functions that depend on them (`AI`, `AO`, `DAQ`, `SCAN`, and `WFM` functions). The following conditions apply:

12-bit E Series Devices	16-bit E Series Devices
• The same set of constants is correct for both polarities of analog input. • One set of constants is valid for unipolar, and another set is valid for bipolar configuration of the analog output channels. When you change the polarity of an analog output channel, NI-DAQ reloads the calibration constants for that channel.	• Calibration constants required by the 16-bit E Series devices for unipolar analog input channels are different from those for bipolar analog input channels. If you are acquiring data from one channel, or if all of the channels you are acquiring data from are configured for the same polarity, NI-DAQ selects the appropriate set of calibration constants for you. If you are scanning several channels, and you mix channels configured for unipolar and bipolar mode in your scan list, NI-DAQ loads the calibration constants appropriate for the polarity that analog input channel 0 is configured for. • Analog output channels on the AT-MIO-16XE-50 can only be configured for bipolar operation. Therefore, NI-DAQ always uses the same constants for the analog output channels.

Config_Alarm_Deadband

Format

status = Config_Alarm_Deadband (deviceNumber, mode, chanStr, trigLevel, deadbandWidth, handle, alarmOnMessage, alarmOffMessage, callbackAddr)

Purpose

Notify NI-DAQ applications when analog input signals meet the alarm-on or alarm-off condition you specified. Notification is done through the NI-DAQ message queue or a callback function that you provide. In Windows and Windows NT, notification is also done through the Windows PostMessage API.

Parameters

Input

Name	**Type**		**Description**
	Windows	**Windows NT**	
deviceNumber	I16	I32	assigned by configuration utility
mode	I16	I32	add or remove high/low alarm events
chanStr	STR	STR	channel string
trigLevel	F64	F64	trigger level in volts
deadbandWidth	F64	F64	the width of the alarm deadband in volts
handle	I16	I32	handle
alarmOnMessage	I16	I32	user-defined alarm-on message
alarmOffMessage	I16	I32	user-defined alarm-off message

Config_Alarm_Deadband

Continued

Parameter Discussion

mode indicates whether to add a new alarm message or remove an old alarm message with the given device.

0:	Add a high alarm deadband event.
1:	Add a low alarm deadband event.
2:	Remove a high alarm deadband event.
3:	Remove a low alarm deadband event.

chanStr is a string description of the trigger analog channel(s) or digital port(s).

The channel string has one of the following formats:

```
xn
SCn!MDn!CHn
AMn!n
```

where

`x:`	AI for analog input channel.
`n:`	Analog channel, digital port, SCXI chassis, SCXI module number, or AMUX-64T device number.
`SC:`	Keyword stands for SCXI chassis.
`MD:`	Keyword stands for SCXI module.
`CH:`	Keyword stands for SCXI channel.
`AM:`	Keyword stands for AMUX-64T device.
`!:`	Delimiter.

For example, the following string specifies onboard analog input channel 5 as the trigger channel:

```
AI5
```

The following string specifies SCXI channel 1 in SCXI module 2 of SCXI chassis 4 as the trigger channel:

```
SC4!MD2!CH1
```

The following specifies AMUX channel 34 on the AMUX-64T device 1 as the trigger channel:

```
AM1!34
```

Config_Alarm_Deadband

Continued

You can also specify more than one channel as the trigger channel by listing all the channels when specifying the channel number. For example, the following string specifies onboard analog input channel 2, 4, 6, and 8 as the trigger channels:

```
AI2,AI4,AI6,AI8
```

Also, if your channel numbers are consecutive, you can use the following shortcut to specify onboard analog input channels 2 through 8 as trigger channels:

```
AI2:8
```

trigLevel is the alarm limit in volts. **trigLevel** and **deadbandWidth** determine the trigger condition.

deadbandWidth specifies, in volts, the hysteresis window for triggering.

handle is the handle to the window you want to receive a Windows message in when **DAQEvent** happens.

alarmOnMessage and **alarmOffMessage** are messages you define. When the alarm-on condition occurs, NI-DAQ passes **alarmOnMessage** back to you. Similarly, when the alarm-off condition occurs, NI-DAQ passes **alarmOffMessage** back to you. The messages can be any value.

In Windows, you can set the message to a value including any Windows predefined messages such as `WM_PAINT`. However, if you want to define your own message, you can use any value ranging from `WM_USER`(0x400) to 0x7fff. This range is reserved by Microsoft for messages you define.

callbackAddr is the address of the user callback function. NI-DAQ calls this function when **DAQEvent** occurs. See `Config_DAQ_Event_Message` for restrictions on this parameter.

Using This Function

To meet the high alarm-on condition, the input signal must first go below (**trigLevel** - **deadbandWidth**/2) volts and then go above (**trigLevel** + **deadbandWidth**/2) volts. On the other hand, to meet the high alarm-off condition, the input signal must first go above (**trigLevel** + **deadbandWidth**/2) volts

Config_Alarm_Deadband

Continued

and then go below (**trigLevel** - **deadbandWidth**/2) volts. See Figure 2-1 for an illustration of the high alarm condition.

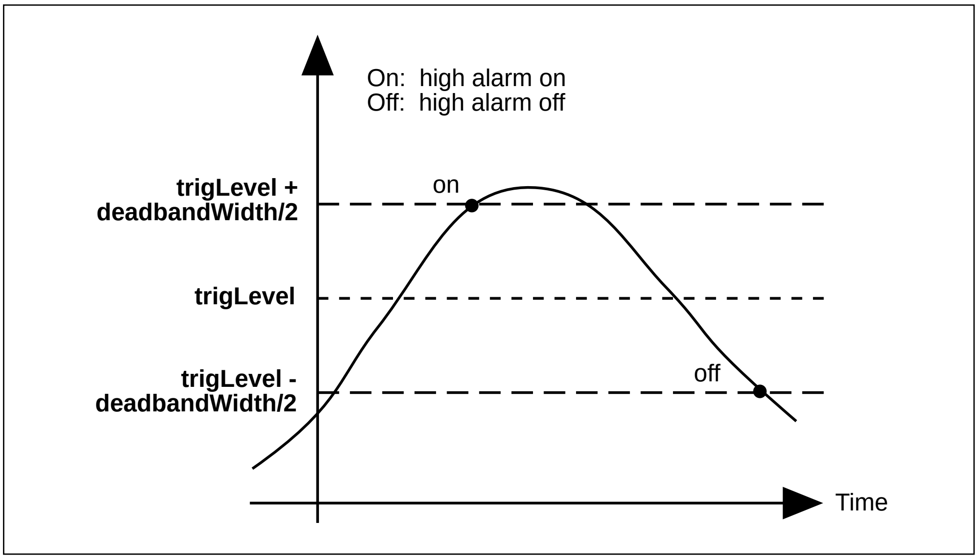


Figure 2-1. High Alarm Deadband

The low alarm deadband trigger condition is the opposite of the high alarm deadband trigger condition. To meet the low alarm-on condition, the input signal must first go above (**trigLevel** + **deadbandWidth**/2) and then go below (**trigLevel** - **deadbandWidth**/2). On the other hand, to meet the low alarm-off condition, the input signal must first go below (**trigLevel** - **deadbandWidth**/2) and then go above (**trigLevel** + **deadbandWidth**/2). See Figure 2-2 for an illustration of the low alarm condition.

Config_Alarm_Deadband

Continued

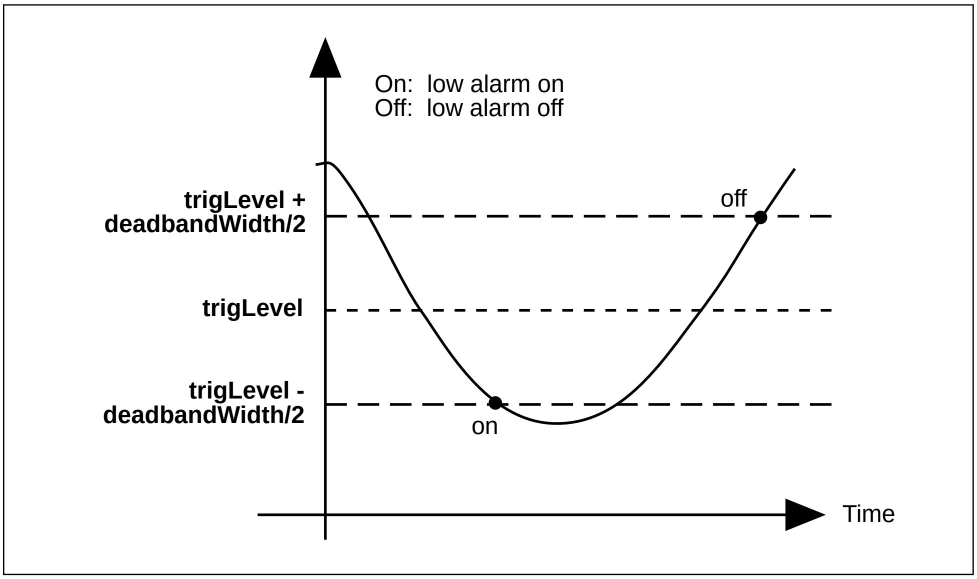


Figure 2-2. Low Alarm Deadband

`Config_Alarm_Deadband` is a high-level function for NI-DAQ event messaging. Because this function uses the current **inputRange** and **polarity** settings to translate **triglevel** and **deadbandWidth** from volts to binary, you should not call `AI_Configure` and change these settings after you have called `Config_Alarm_Deadband`. For more information on NI-DAQ event messaging, see the low-level function `Config_DAQ_Event_Message`. When you are using this function, the analog input data acquisition must be run with interrupts only (programmed I/O mode). You cannot use DMA. See `Set_DAQ_Device_Info` for how to change modes.

Config_ATrig_Event_Message

Format

status = Config_ATrig_Event_Message (deviceNumber, mode, chanStr, trigLevel, windowSize, trigSlope, trigSkipCount, pretrigScans, postTrigScans, handle, message, callbackAddr)

Purpose

Notify NI-DAQ applications when the trigger channel signal meets certain criteria you specify. Notification is done through the NI-DAQ message queue and/or callback function you provide. In Windows and Windows NT, notification is also done through the Windows PostMessage API.

Parameters

Input

Name	Type		Description
	Windows	Windows NT	
deviceNumber	I16	I32	assigned by configuration utility
mode	I16	I32	add or remove a message
chanStr	STR	STR	channel string
trigLevel	F64	F64	trigger level in volts
windowSize	F64	F64	hysteresis window size in volts
triggerSlope	I16	I32	trigger slope
trigSkipCount	U32	U32	number of triggers
preTrigScans	U32	U32	number of scans to skip before trigger event
postTrigScans	U32	U32	number of scans after trigger event

Config_ATrig_Event_Message

Continued

Name	Type		Description
	Windows	Windows NT	
handle	I16	I32	handle
message	I16	I32	user-defined message
callbackAddr	U32	U32	user callback function address

Parameter Discussion

mode indicates whether to add a new alarm message or to remove an old alarm message with the given device.

0: Remove an existing analog trigger event.
1: Add a new analog trigger event.

chanStr is a string description of the trigger analog channel(s) or digital port(s).

The channel string has one of the following formats:

```
xn
SCn!MDn!CHn
AMn!n
```

where

`x`:	`AI` for analog input channel.
`n`:	Analog channel, digital port, SCXI chassis, SCXI module number, or AMUX-64T device number.
`SC`:	Keyword stands for SCXI chassis.
`MD`:	Keyword stands for SCXI module.
`CH`:	Keyword stands for SCXI channel.
`AM`:	Keyword stands for AMUX-64T device.
`!`:	Delimiter.

For example, the following string specifies an onboard analog input channel 5 as the trigger channel:

```
AI5
```

Config_ATrig_Event_Message

Continued

The following string specifies SCXI channel 1 in SCXI module 2 of SCXI chassis 4 as the trigger channel:

```
SC4!MD2!CH1
```

The following specifies AMUX channel 34 on the AMUX-64T device 1 as the trigger channel:

```
AM1!34
```

You can also specify more than one channel as the trigger channel by listing all the channels when specifying channel number. For example, the following string specifies onboard analog input channel 2, 4, 6, and 8 as the trigger channels:

```
AI2,AI4,AI6,AI8
```

Also, if your channel numbers are consecutive, you can use the following shortcut to specify onboard analog input channels 2 through 8 as trigger channels:

```
AI2:8
```

trigLevel is the alarm limit in volts. **trigLevel** and **windowSize** determine the trigger condition.

windowSize is the number of volts below **trigLevel** for positive slope or above the analog trigger level for negative slope that the input signal must go before NI-DAQ recognizes a valid trigger crossing at the analog trigger level.

trigSlope is the slope the input signal should trigger on.

0:	Trigger on either positive and negative slope.
1:	Trigger on positive slope.
2:	Trigger on negative slope.

trigSkipCount is the number of valid triggers NI-DAQ ignores. It can be any value greater than or equal to zero. For example, if **trigSkipCount** is 3, you will be notified when the fourth trigger occurs.

preTrigScans is the number of scans of data NI-DAQ will collect before looking for the very first trigger. Setting **preTrigScans** to 0 will cause NI-DAQ to look for the first trigger as soon as the DAQ process begins.

Config_ATrig_Event_Message

Continued

postTrigScans is the number of scans of data NI-DAQ will collect after the **trigSkipCount** triggers before notifying you.

handle is the handle to the window you want to receive a Windows message in when **DAQEvent** happens.

message is a message you define. When **DAQEvent** happens, NI-DAQ passes **message** back to you. **message** can be any value.

In Windows, you can set **message** to a value including any Windows predefined messages (such as `WM_PAINT)`. However, if you want to define your own message, you can use any value ranging from `WM_USER`(0x400) to 0x7fff. This range is reserved by Microsoft for messages you define.

callbackAddr is the address of the user callback function. NI-DAQ calls this function when **DAQEvent** occurs. See `Config_DAQ_Event_Message` for restrictions on this parameter.

Config_ATrig_Event_Message

Continued

Using This Function

To meet the positive trigger condition, the input signal must first go below (**trigLevel** - **windowSize**) and then go above **trigLevel**. On the other hand, to meet the negative trigger condition, the input signal must first go above (**trigLevel** + **windowSize**) and then go below **trigLevel**. Figure 2-3 shows these conditions.

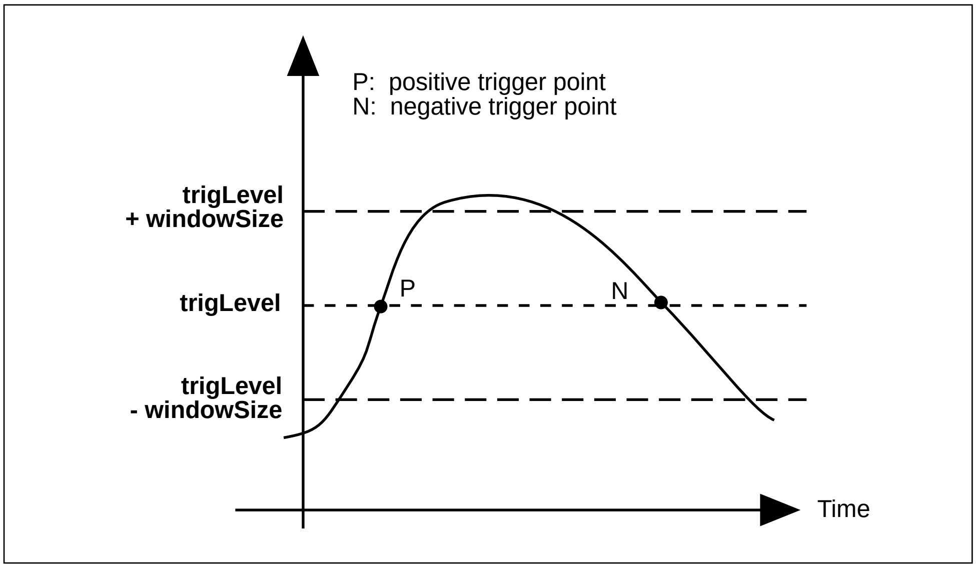


Figure 2-3. Analog Trigger Event

`Config_ATrig_Event_Message` is a high-level function for NI-DAQ event messaging. Because this function uses the current **inputRange** and **polarity** settings to translate **trigLevel** and **windowSize** into binary units, you should not call `AI_Configure` and change these settings after you have called `Config_ATrig_Event_Message`. For more information on NI-DAQ event messaging, see the low-level function `Config_DAQ_Event_Message`. When you are using this function, the analog input data acquisition must be run with just interrupts (programmed I/O mode). You cannot use DMA. See `Set_DAQ_Device_Info` for how to change modes.

Config_DAQ_Event_Message

Format

status = Config_DAQ_Event_Message (deviceNumber, mode, chanStr, DAQEvent, DAQTrigVal0, DAQTrigVal1, trigSkipCount, preTrigScans, postTrigScans, handle, message, callbackAddr)

Purpose

Notify NI-DAQ applications when the status of an asynchronous DAQ operation (initiated by a call to `DAQ_Start`, `DIG_Block_Out`, and so on) meets certain criteria you specify. Notification is done through the Windows PostMessage API. In Windows 3.1, you can also use a callback function.

If your NI-DAQ application does not require high-speed data transfer, the NI-DAQ event messaging system can offer you more **DAQEvent** options. **DAQEvents** 3 through 8 are designed for interrupt-driven data acquisition. NI-DAQ evaluates **DAQEvent** on every data point during run time.

Parameters

Input

Name	Type		Description
	Windows	Windows NT	
deviceNumber	I16	I32	assigned by configuration utility
mode	I16	I32	add or remove a message
chanStr	STR	STR	channel string
DAQEvent	I16	I32	event criteria
DAQTrigVal0	I32	I32	general-purpose trigger value
DAQTrigVal1	I32	I32	general-purpose trigger value
trigSkipCount	U32	U32	number of triggers to skip

Config_DAQ_Event_Message

Continued

Name	Type		Description
	Windows	Windows NT	
preTrigScans	U32	U32	number of scans before trigger event
postTrigScans	U32	U32	number of scans after trigger event
handle	I16	I32	handle
message	I16	I32	user-defined message
callbackAddr	U32	U32	user callback function address

Parameter Discussion

mode indicates whether to add a new message, remove an old message, or clear all messages associated with the given device.

0:	Clear all messages associated with the device including messages configured with `Config_Alarm_Deadband` and `Config_ATrig_Event_Message`.
1:	Add a new message.
2:	Remove an existing message.

chanStr is a string description of the trigger analog channel(s) or digital port(s).

The channel string has one of the following formats:

```
xn
SCn!MDn!CHn
AMn!n
```

where

`x`: `AI` for analog input channel.
`DI` for digital input channel.
`DO` for digital output channel.
`CTR` for counter.
`EXT` for external timing input.

Config_DAQ_Event_Message

Continued

`n`:	Analog channel, digital port, counter, SCXI chassis, SCXI module number, or AMUX-64T device number
`SC`:	Keyword stands for SCXI chassis.
`MD`:	Keyword stands for SCXI module.
`CH`:	Keyword stands for SCXI channel.
`AM`:	Keyword stands for AMUX-64T device.
`!`:	Delimiter.

For example, the following string specifies an onboard analog input channel 5 as the trigger channel:

```
AI5
```

When using messaging with an SCXI module in Parallel mode, you must refer to the channels by their onboard channel numbers, not their SCXI channel numbers.

The following string specifies SCXI channel 1 in SCXI module 2 of SCXI chassis 4 as the trigger channel:

```
SC4!MD2!CH1
```

The following specifies AMUX channel 34 on the AMUX-64T device 1 as the trigger channel:

```
AM1!34
```

You can specify only one AMUX channel in the **chanStr** parameter.

You can also specify more than one channel as the trigger channel by listing all the channels when specifying channel number. For example, the following string specifies onboard analog input channel 2, 4, 6, and 8 as the trigger channels:

```
AI2,AI4,AI6,AI8
```

Also, if your channel numbers are consecutive, you can use the following shortcut to specify onboard analog input channels 2 through 8 as trigger channels:

```
AI2:8
```

DAQEvent indicates the event criteria for user notification. The following table describes the different types of messages available in NI-DAQ. A *scan* is defined as one pass through all the analog input channels or digital ports that are part of your asynchronous DAQ operation.

Config_DAQ_Event_Message

Continued

☞ **Note:** ***DAQEvent****=7, 8, and 9 are not available in Windows NT.*

DAQEvent*=7 and 8 are not available when using an SCXI-1200 with Remote SCXI.*

If you are using a DAQ device in a Remote SCXI configuration for digital I/O operations, DAQ events are not supported.

Table 2-3. DAQ Event Messages

DAQEvent Type	Code	Description of Message	Usable devices <IRQ-based operation>
Acquire or generate *N* scans	0	Send exactly one message when an asynchronous operation has completed **DAQTrigVal0** scans.	MIO and AI devices <AI> Lab and 1200 series devices <AI, DIO> 516 and LPM devices, DAQCard-500/700 AT-DIO-32F <DIO> PC-DIO-24, PC-DIO-96/PnP
Every *N* scans	1	Send a message each time an asynchronous operation completes a multiple of **DAQTrigVal0** scans. **chanStr** indicates the type of channel or port, but the actual channel or port number is ignored.	MIO and AI devices <AI> Lab and 1200 series devices <AI, DIO> 516 and LPM devices, DAQCard-500/700 AT-DIO-32F <DIO> PC-DIO-24, PC-DIO-96/PnP

Config_DAQ_Event_Message

Continued

Table 2-3. DAQ Event Messages (Continued)

DAQEvent Type	Code	Description of Message	Usable devices <IRQ-based operation>
Completed operation or stopped by error	2	Send exactly one message when an asynchronous operation completes or is stopped for an error. **chanStr** indicates the type of channel or port, but the actual channel or port number is ignored.	MIO and AI devices <AI> Lab and 1200 series devices <AI, DIO> 516 and LPM devices, DAQCard-500/700 AT-DIO-32F <DIO> PC-DIO-24, PC-DIO-96/PnP
Voltage out of bounds	3	Send a message each time a data point from any channel in **chanStr** is outside of the voltage region bounded by **DAQTrigVal0** and **DAQTrigVal1**, where **DAQTrigVal0** ≥ **DAQTrigVal1**.	MIO and AI devices <AI> Lab and 1200 series devices <AI> 516 and LPM devices, DAQCard-500/700
Voltage within bounds	4	Send a message each time a data point from any channel in **chanStr** is inside of the voltage region bounded by **DAQTrigVal0** and **DAQTrigVal1**, where **DAQTrigVal0** ≥ **DAQTrigVal1**.	MIO and AI devices <AI> Lab and 1200 series devices <AI> 516 and LPM devices, DAQCard-500/700

Config_DAQ_Event_Message

Continued

Table 2-3. DAQ Event Messages (Continued)

DAQEvent Type	Code	Description of Message	Usable devices <IRQ-based operation>
Analog positive slope triggering	5	Send a message when data from any channel in **chanStr** positively triggers on the hysteresis window specified by **DAQTrigVal0** and **DAQTrigVal1**, where **DAQTrigVal0** ≥ **DAQTrigVal1**. To positively trigger, data must first go below **DAQTrigVal1** and above **DAQTrigVal0**.	MIO and AI devices <AI> Lab and 1200 series devices <AI> 516 and LPM devices, DAQCard-500/700
Analog negative slope triggering	6	Send a message when data from any channel in **chanStr** negatively triggers on the hysteresis window specified by **DAQTrigVal0** and **DAQTrigVal1**, where **DAQTrigVal0** ≥ **DAQTrigVal1**. To negatively trigger, data must first go above **DAQTrigVal0** and below **DAQTrigVal1**.	MIO and AI devices <AI> Lab and 1200 series devices <AI> 516 and LPM devices, DAQCard-500/700
Digital pattern not matched	7	Send a message when data from any digital port in **chanStr** causes this statement to be true: data AND **DAQTrigVal0** NOT EQUAL **DAQTrigVal1**. Only the lower word is relevant.	Lab and 1200 series devices (except an SCXI-1200 with Remote SCXI) <DIO> PC-DIO-24, PC-DIO-96/PnP AT-MIO-16D <DIO>

Config_DAQ_Event_Message

Continued

Table 2-3. DAQ Event Messages (Continued)

DAQEvent Type	Code	Description of Message	Usable devices <IRQ-based operation>
Digital pattern matched	8	Send a message when data from any digital port in **chanStr** causes this statement to be true: data AND **DAQTrigVal0** EQUAL **DAQTrigVal1**. Only the lower word is relevant	Lab and 1200 series devices (except an SCXI-1200 with Remote SCXI) <DIO> PC-DIO-24, PC-DIO-96/PnP AT-MIO-16D <DIO>
Counter pulse event	9	Send a message each time a pulse occurs in a timing signal. You can configure only one such event message at a time on a device, except on the PC-TIO-10, which can have two. (Refer to Note)	Am9513-based devices PC-TIO-10

DAQEvent=3 through 8—These **DAQEvents** are for interrupt-driven data acquisition only. See `Set_DAQ_Device_Info` for switching between interrupt-driven and DMA-driven data acquisition.

If you are using a Lab or 1200 Series device in pretrigger mode, NI-DAQ will not send any messages you configure for the end of the acquisition. These devices do not generate an interrupt at the end of the acquisition when in pretrigger mode.

DAQEvent=9—NI-DAQ sends a message when a transition (low to high or high to low) appears on a counter output or external timing signal I/O pin. Table 2-3 shows the possible counters and external timing signals that are valid for each supported device.

Config_DAQ_Event_Message

Continued

Table 2-4. Valid Counters and External Timing Signals for DAQEvent = 9

Data Acquisition Device	I/O Pin	I/O Pin State Change
AT-MIO-16	OUT2	low to high
AT-MIO-16D	OUT2	low to high
AT-MIO-16F-5	OUT1, OUT2, OUT5, or EXTDACUPDATE*	high to low
AT-MIO-16X	OUT1, OUT2, OUT5, or EXTTMRTRIG*	high to low
AT-MIO-64F-5	OUT1, OUT2, OUT5, or EXTTMRTRIG*	high to low
PC-TIO-10	EXTIRQ1 or EXTIRQ2	high to low

If you are using one of the counters on the PC-TIO-10 for your timing signal, you must connect the counter output to the EXTIRQ pin either externally through the I/O connector or with the two jumpers on the device. The jumpers connect the OUT2 and OUT7 pins with the EXTIRQ1 and EXTIRQ2 pins, respectively. NI-DAQ returns an error if you specify a counter that is in use. You should use EXT1 for the **chanStr** parameter regardless of which EXTIRQ pin you are using. The PC-TIO-10 can have two of these event messages configured at the same time, therefore you must specify which pin you want to use on the PC-TIO-10 with the **DAQTrigVal0** parameter.

To use **DAQEvent** = 9, you must configure the device for interrupt-driven waveform generation. This **DAQEvent** works by using the waveform generation timing system. Thus, you cannot use waveform generation or single point analog output with delayed update mode and this **DAQEvent** at the same time on the same device. Also, **DAQEvent** = 9 is not valid for the E Series devices.

trigSkipCount is the number of valid triggers NI-DAQ ignores. It can be any value greater than or equal to zero. For example, if **trigSkipCount** is 3, NI-DAQ will notify you when the fourth trigger occurs.

Config_DAQ_Event_Message

Continued

preTrigScans is the number of scans of data NI-DAQ will collect before looking for the very first trigger. Setting **preTrigScans** to 0 will cause NI-DAQ to look for the first trigger as soon as the DAQ process begins.

postTrigScans is the number of scans of data NI-DAQ will collect after the **trigSkipCount** triggers before notifying you. Setting **postTrigScans** to 0 will cause event notification to happen as soon as the trigger occurs.

Refer to the following table for further details on usable parameters for each **DAQEvent** type.

Table 2-5. Usable Parameters for Different DAQ Events Codes

Parameter	**DAQEvent**									
	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
chanStr (where *n* and *m* are numbers)	AI*n*, DI*n*, DO*n*, SC*n*!..., AM*n*!*m*	AI*n*, DI*n*, DO*n*, SC*n*!..., AM*n*!*m*	AI*n*, DI*n*, DO*n*, SC*n*!..., AM*n*!*m*	AI*n*, SC*n*!..., AM*n*!*m*	AI*n*, SC*n*!..., AM*n*!*m*	AI*n*, SC*n*!..., AM*n*!*m*	AI*n*, SC*n*!..., AM*n*!*m*	DI*n*, DO*n*	DI*n*, DO*n*	CTR*n*, EXT1
DAQTrigVal0	no. of scans, must be greater than 0	no. of scans, must be greater than 0 (see note below)	ignored	upper bound for analog alarm region (binary), must be greater than or equal to **DAQTrigVal1**	upper bound for analog alarm region (binary), must be greater than or equal to **DAQTrigVal1**	upper bound for hysteresis window (binary), must be greater than or equal to **DAQTrigVal1**	upper bound for hysteresis window (binary), must be greater than or equal to **DAQTrigVal1**	digital pattern mask (decimal)	digital pattern mask (decimal)	EXTIRQ no. (1 or 2) if **chanStr** = EXT*n* for the PC-TIO-10, otherwise ignored
DAQTrigVal1	ignored	ignored	ignored	lower bound for analog alarm region (binary)	lower bound for analog alarm region (binary)	lower bound for hysteresis window (binary)	lower bound for hysteresis window (binary)	digital pattern not to match (decimal)	digital pattern to match (decimal)	ignored
trigSkipCount	ignored	ignored	ignored	ignored	ignored	no. of triggers to skip	no. of triggers to skip	ignored	ignored	ignored
preTrigScans	ignored	ignored	ignored	ignored	ignored	no. of scans before trigger condition is met	no. of scans before trigger condition is met	ignored	ignored	ignored
postTrigScans	ignored	ignored	ignored	ignored	ignored	no. of scans after trigger condition is met	no. of scans after trigger condition is met	ignored	ignored	ignored

Config_DAQ_Event_Message

Continued

For the parameters that are ignored, set them to 0.

For **DAQEvent** = 1, **DAQTrigVal0** must be greater than zero. If you are using DMA with double buffers or a pretrigger data acquisition, **DAQTrigVal0** must be an even divisor of the size of the buffer in scans.

handle is the handle to the window you want to receive a Windows message in when **DAQEvent** happens.

message is a message you define. When **DAQEvent** happens, NI-DAQ passes **message** back to you. **message** can be any value.

You can set **message** to a Windows predefined message such as `WM_PAINT`. You can also define your own message by using any value ranging from `WM_USER` (0x0400) to 0x7fff. This range is reserved by Microsoft for messages you define.

callbackAddr is the address of the user callback function. If you are programming in Windows 3.1 using C or C++, or using LabWindows/CVI, NI-DAQ calls this function when **DAQEvent** occurs. To create a proper callback address for Windows 3.1 using C or C++ (excluding LabWindows/CVI), use the `MakeProcInstance` function. When you are finished with the callback function, free the callback address with `FreeProcInstance`.

If you are using LabWindows/CVI, simply specify the name of your callback function.

If you are going to check the NI-DAQ message queue for events or are using NI-DAQ for Windows 95 or Windows NT, set this parameter to zero because you do not need to provide a callback function.

Using This Function

This function is designed to notify your application when **DAQEvent** occurs. Using **DAQEvents** eliminates continuous polling of data acquisition processes through NI-DAQ functions.

For example, if you have a double-buffered DAQ application, instead of calling `DAQ_DB_HalfReady` continuously, you can call `Config_DAQ_Event_Message` and set **DAQEvent** to 1 and **DAQTrigVal0** to be one-half your buffer size in units of scans. Then, NI-DAQ will notify your application when it is time to call `DAQ_DB_Transfer`. The same concept applies to digital input/output block functions.

Config_DAQ_Event_Message

Continued

☞ **Note:** ***NI-DAQ does not notify your application if your DAQ buffer resides in AT-DSP2200 onboard memory.***

To define a message, call `Config_DAQ_Event_Message` before starting your DAQ process. You can associate more than one message to the same device by calling `Config_DAQ_Event_Message` as many times as you need to. For example, NI-DAQ can notify you when data is in an invalid/alarm region (**DAQEvent** 3 through 6) or when NI-DAQ collects **DAQTrigVal0** scans of data (**DAQEvent** 0).

After you define a message, it remains active until you call `Init_DA_Brds` or `Config_DAQ_Event_Message` to remove messages. To remove a message, call `Config_DAQ_Event_Message` with **mode** set to 2 for removing a specific message. When removing a message, make sure to provide all the information defining the message, such as **chanStr(SCXIchassisID**, **moduleSlot**, **chanType**, **chan)**, **DAQEvent**, **DAQTrigVal0**, **DAQTrigVal1**, **trigSkipCount**, **preTrigScans**, **postTrigScans**, **handle**, **message**, and **callbackAddr**.

To remove all messages associated with the device, call `Config_DAQ_Event_Message` with **mode** set to zero and with all other arguments except **deviceNumber** set to zero.

In NI-DAQ for Windows 3.1, Windows 95, and Windows NT applications, the notification is done through the Windows API function PostMessage. When any trigger event happens, NI-DAQ calls PostMessage as follows:

```
int PostMessage  (HWND handle, UINT message, WPARAM wParam,
                  LPARAM lParam)
```

handle and **message** are the same handle and message as previously defined. The lower byte of **wParam** is the device and the upper byte of **wParam** is a boolean flag, **doneFlag**, indicating whether the DAQ process has ended.

doneFlag = 0: Data acquisition is still running.
doneFlag = 1: Data acquisition has stopped.

lParam contains the number of the scan in which **DAQEvent** occurred.

How soon your application receives the message depends on how often your application calls the Windows API function GetMessage and what other messages are in the Windows message queue.

Config_DAQ_Event_Message

Continued

For example, if you are programming with Visual C++ or Borland C++ in Windows, you may have something like the following in your `WinMain` routine:

```
while (GetMessage (&msg, NULL, 0, 0))
{
      TranslateMessage(&msg);
      DispatchMessage(&msg);
}
```

You will most likely have registered a message handling routine in your Windows application. In that routine, you can handle the message as follows:

```
LRESULT CALLBACK WindowProc(HWND hWnd, UINT uMsgId, WPARAM wParam, LPARAM
lParam)
{
      static unsigned long int uNIDAQeventCount = 0;
      short DAQeventDevice;
      short doneFlag;
      long scansDone;
      switch (uMsgId)
      {
            case WM_PAINT:
                  //..handle this message...
                  break;
            case WM_DESTROY:
                  //..handle this message...
                  break;
            case WM_NIDAQ_MSG:
                  //****************************************
                  //put your NI-DAQ Message handling here!
                  //****************************************

                  // increment static counter
                  uNIDAQeventCount++;
                  DAQeventDevice = (wParam & 0x00FF);
                  doneFlag = (wParam & 0xFF00) >> 8;
                  scansDone = lParam;

                  //..handle this message...
                  return 0;
                  break;
            default:
```

Config_DAQ_Event_Message

Continued

```
                    // handle other usual messages...
                    return DefWindowProc (hWnd, uMsgId, wParam, lParam);
        }
}
```

Windows 3.1 Callback Functions

Notification is also done through a call to your own callback function.

To enable the callback function, you need to provide the address of the callback routine. Therefore, you must write your application in programming languages that support function pointers, such as C and Assembly.

☞ **Note:** ***If your programming language does not support function pointers, you can call*** `Get_DAQ_Event` ***and*** `Peek_DAQ_Event` ***to check for events. See*** `Get_DAQ_Event` ***and*** `Peek_DAQ_Event` ***for further information.***

If you choose to use a callback function, it must have the following prototype:

```
void _loadds Callback (HWND handle, UINT message, WPARAM wParam,
                       LPARAM lParam)
```

handle and **message** are the same handle and message as previously defined. The lower byte of **wParam** is the device and the upper byte of **wParam** is a boolean flag, **doneFlag**, indicating whether the DAQ process has ended. In Windows 95 and Windows NT, `int` is 4 bytes long and the high word is not used.
doneFlag = 0: Data acquisition is still running.
doneFlag = 1: Data acquisition has stopped.

lParam contains the number of the scan in which **DAQEvent** occurred.

If you are using LabWindows/CVI, your callback function is called by means of messaging. No special precautions are required. However, you do need to pay special attention when writing a callback function in Windows, since it is a call during interrupt time.

If you are writing a Windows 3.1 interrupt callback function using C or C++ (excluding LabWindows/CVI),

- Put your ISR code in a DLL.

Config_DAQ_Event_Message

Continued

- Mark your callback as `FAR PASCAL` function. For example, void `FAR PASCAL NIDAQInterruptCallback`(...).
- Add your callback function in the `EXPORTS` statement in your module definition file.
- Mark the callback function's code segment as `FIXED` in the `CODE` statement of your module-definition file.
- Mark all the global data to be accessed by the callback function as `FIXED` in the `DATA` statement of your module-definition file.

In the callback functions, do not make calls to functions that are not re-entrant. Most Windows API functions are not re-entrant and will not work properly in callback functions.

The following information is an excerpt from the Microsoft Developer Network Library CD. Your ISR may call only the following Windows API functions at interrupt time:

- `PostMessage`
- `PostAppMessage`

and only the following multimedia system functions:

- `timeGetSystemTime`
- `timeGetTime`
- `timeSetEvent`
- `timeKillEvent`
- `midiOutShortMsg`
- `midiOutLongMsg`
- `OutputDebugStr` (this function exists only in the debugging version of the MMSYSTEM library; this is not the same function as `OutputDebugString`)

Because the callback function is called at actual hardware interrupt time, the time it takes to respond to the triggered DAQ Event will most likely be shorter than polling for messages with the Windows API function `GetMessage`. However, remember that the callback function feature is only supported in Windows 3.1.

Configure_HW_Analog_Trigger

Format

status = Configure_HW_Analog_Trigger (deviceNumber, onOrOff, lowValue, highValue, mode, trigSource)

Purpose

Configures the hardware analog trigger. The hardware analog triggering circuitry produces a digital trigger that you can use for any of the signals available through the `Select_Signal` function by selecting **source** = `ND_PFI_0` (AT-AI-16XE-10, AT-MIO-16E-1, AT-MIO-16E-2, AT-MIO-16XE-10, AT-MIO-64E-3, DAQCard-AI-16E-4, NEC-AI-16E-4, and NEC-MIO-16E-4, and only).

☞ **Note:** ***NI-DAQ for Windows NT does not support this function.***

Parameters

Input

Name	Type		Description
	Windows	Windows NT	
deviceNumber	I16	n/a	assigned by configuration utility
onOrOff	U32	n/a	turns the analog trigger on or off
lowValue	I32	n/a	specifies the low level used for analog triggering
highValue	I32	n/a	specifies the high level used for analog triggering
mode	U32	n/a	the way the triggers are generated
trigSource	U32	n/a	the source of the signal used for triggering

Configure_HW_Analog_Trigger

Continued

Parameter Discussion

Legal ranges for the **onOrOff**, **mode**, and **trigSource** parameters are given in terms of constants that are defined in a header file. The header file you should use depends on the language you are using:

- C programmers—`NIDAQCNS.H` (`DATAACQ.H` for LabWindows/CVI)
- BASIC programmers—`NIDAQCNS.INC` (Visual Basic for Windows programmers should refer to the *Programming Language Considerations* section in Chapter 1 for more information.)
- Pascal programmers—`NIDAQCNS.PAS`

Use **onOrOff** to inform NI-DAQ whether you want to turn the analog trigger on or off. Legal values for this parameter are `ND_ON` and `ND_OFF`.

Use **lowValue** and **highValue** to specify the levels you want to use for triggering. The legal range for the two values is 0 to 255 (0–4,095 for the AT-AI-16XE-10 and AT-MIO-16XE-10). In addition, **lowValue** must be less than **highValue**. The voltage levels corresponding to **lowValue** and **highValue** are as follows:

- When **trigSource** = `ND_PFI_0`, 0 corresponds to -10 V and 255 (4,095 for the AT-AI-16XE-10 and AT-MIO-16XE-10) corresponds to +10 V; values between 0 and 255 (4,095 for the AT-AI-16XE-10 and AT-MIO-16XE-10) are evenly distributed between -10 V and +10 V. You can use `ND_PFI_0` as the analog signal you are triggering off of at the same time you designate `ND_PFI_0` as a source for a `Select_Signal` **signal**.
- When **trigSource** = `ND_THE_AI_CHANNEL` and the channel is in bipolar mode, 0 corresponds to -5 V, 255 (4,095 for the AT-AI-16XE-10 and AT-MIO-16XE-10) corresponds to +5 V; values between 0 and 255 (4,095 for the AT-AI-16XE-10 and AT-MIO-16XE-10) are evenly distributed between -5 V and +5 V.
- When **trigSource** = `ND_THE_AI_CHANNEL` and the channel is in unipolar mode, 0 corresponds to 0 V, 255 (4,095 for the AT-AI-16XE-10 and AT-MIO-16XE-10) corresponds to +10 V; values between 0 and 255 (4,095 for the AT-AI-16XE-10 and AT-MIO-16XE-10) are evenly distributed between 0 V and +10 V.

The end of this section contains an example calculation for **lowValue**.

Use the **mode** parameter to tell NI-DAQ how you want analog triggers to be converted into digital triggers that the onboard hardware can use for timing.

Configure_HW_Analog_Trigger

Continued

The following paragraphs and figures show all of the available modes and illustrations of corresponding trigger generation scenarios. Values specified by **highValue** and **lowValue** are represented using dashed lines, and the signal used for triggering is represented using a solid line.

- `ND_BELOW_LOW_LEVEL`—The trigger is generated when the signal value is less than the **lowValue**. **highValue** is unused.

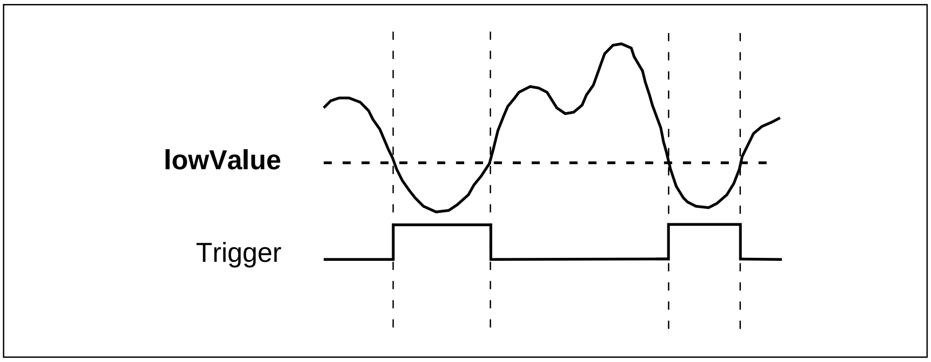


Figure 2-4. ND_BELOW_LOW_LEVEL

- `ND_ABOVE_HIGH_LEVEL`—The trigger is generated when the signal value is greater than the **highValue**. **lowValue** is unused.

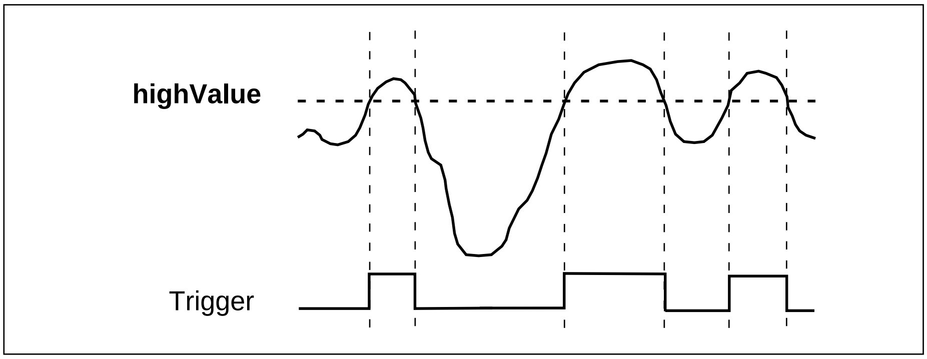


Figure 2-5. ND_ABOVE_HIGH_LEVEL

Configure_HW_Analog_Trigger

Continued

- `ND_INSIDE_REGION`—The trigger is generated when the signal value is between the **lowValue** and the **highValue**.

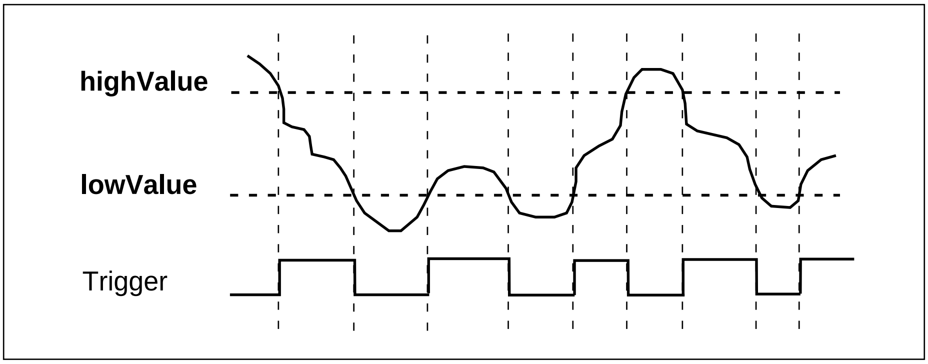


Figure 2-6. ND_INSIDE_REGION

- `ND_HIGH_HYSTERESIS`—The trigger is generated when the signal value is greater than the **highValue**, with hysteresis specified by **lowValue**.

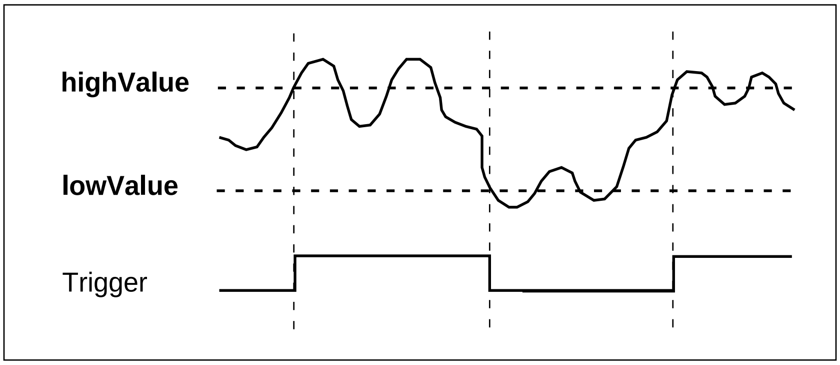


Figure 2-7. ND_HIGH_HYSTERESIS

Configure_HW_Analog_Trigger

Continued

- ND_LOW_HYSTERESIS—The trigger is generated when the signal value is less than the **lowValue**, with hysteresis specified by **highValue**.

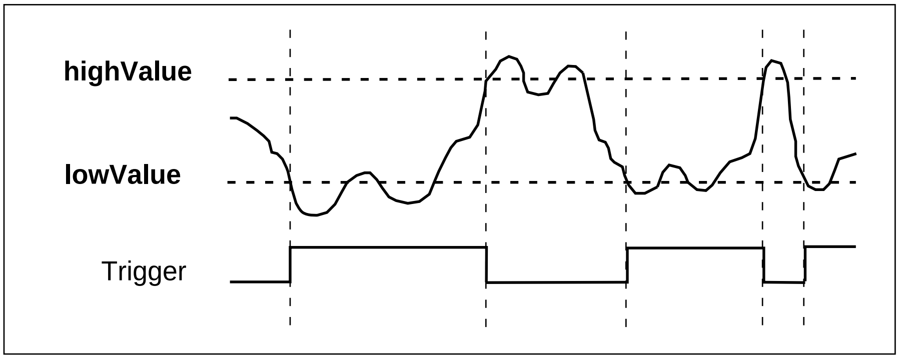


Figure 2-8. ND_LOW_HYSTERESIS

Use the **trigSource** parameter to specify the source of the trigger you want to use. The legal values are ND_PFI_0 and ND_THE_AI_CHANNEL.

Set **trigSource** to ND_PFI_0 if you want the trigger to come from the PFI0/TRIG1 pin. You need to connect the analog signal you want to use for triggering to the PFI0/TRIG1 pin. If you want to generate triggers based on an analog signal that takes a wide range of values between -10 V and +10 V, you should use this setting.

You should select ND_THE_AI_CHANNEL for **trigSource** only if you want to generate triggers based on a low-range analog signal, if you are concerned about signal quality and are using a shielded cable, or if you want the trigger to be based on an analog input channel in the differential mode. Using this selection is non-trivial.

Configure_HW_Analog_Trigger

Continued

If you set **trigSource** to `ND_THE_AI_CHANNEL`, you will be able to use the signal connected to one of the analog input pins for triggering. In this case, the signal will be amplified on the device before it is used for trigger generation. You can use this source selection under the following conditions:

- You want to perform data acquisition from a single analog input channel (the `DAQ` family of functions). You can only use the channel you are acquiring from for analog triggering.
- You want to perform data acquisition from more than one analog input channel (a combination of the `DAQ` and `SCAN` families of functions). The only analog input channel you can use as the start trigger is the first channel from your list of channels. You cannot use this form of the analog trigger for the stop trigger in case of pretriggered data acquisition.
- You do not want to perform any analog input operations (the `AI`, `DAQ`, and `SCAN` families of functions). You must use `AI_Setup` to select the analog input channel you want to use and the gain of the instrumentation amplifier. You can also use `AI_Configure` to alter the configuration of the analog input channel.
- You want to use `AI_Check`, and you want to use the analog trigger for conversion timing. You do not have to perform any special steps.

The reason for these constraints is that if you are scanning among several analog input channels, signals from those channels are multiplexed in time, and the analog triggering circuitry is unable to distinguish between signals from individual channels in this case.

Using This Function

When you use this function, you activate the onboard analog triggering hardware. This onboard hardware generates a digital trigger that the DAQ-STC then uses for timing and control. To use the analog trigger, you need to use this function and the `Select_Signal` function. If you want to use analog triggering, use as much hysteresis as your application will allow because the circuitry used for this purpose is very noise-sensitive.

When you use `Select_Signal`, set **source** to `ND_PFI_0` for your signal, and set **polarity** as appropriate. Notice that the two polarity selections give you timing control in addition to the five triggering modes listed here.

Configure_HW_Analog_Trigger

Continued

For example, if you set **source** to `ND_THE_AI_CHANNEL`, the channel you are interested in is in bipolar mode, you want a gain of 100, and you want to set the voltage window for triggering to +35 mV and +45 mV for your original signal (that is, signal before amplification by the onboard amplifier), you should make the following programming sequence:

status = Configure_HW_Analog_Trigger (deviceNumber, `ND_ON`**, 218, 243, mode,** `ND_THE_AI_CHANNEL`**).**

Status = Select_Signal (deviceNumber, `ND_IN_START_TRIGGER`**,** `ND_PFI_0`**,** `ND_LOW_TO_HIGH`**)**

To calculate **lowValue** in the previous example, do the following:

1. Multiply 35 mV by 100 to adjust for the gain to get 3.5 V.
2. Use the following formula to map the 3.5 V from the -5 V to +5 V scale to a value on the 0 to 255 (0–4,095 for the AT-AI-16XE-10 and AT-MIO-16XE-10) scale:

   value = (3.5/5 + 1) * 128 = 218

In general, the scaling formulas are as follows:

- For an analog input channel in the bipolar mode:value = (voltage/5 + 1) * 128
- For an analog input channel in the unipolar mode:value = (voltage/10) * 256
- For the PFI0/TRIG1 pin:value = (voltage/10 + 1) * 128

If you apply any of the formulas and get a value equal to 256, use the value 255 instead; if you get 4,096 with the AT-AI-16XE-10 or AT-MIO-16XE-10, use 4,095 instead.

The following programming sequence could be used to set up an acquisition to be triggered using the hardware analog trigger, where the trigger source is the PFI0/TRIG1 pin:

status = Configure_HW_Analog_Trigger (deviceNumber, `ND_ON`**, lowValue, highValue, mode,** `ND_PFI_0`**)**

status = Select_Signal (deviceNumber, `ND_IN_START_TRIGGER`**,** `ND_PFI_0`**,** `ND_LOW_TO_HIGH`**)**

CTR_Config

Format

status = CTR_Config (deviceNumber, ctr, edgeMode, gateMode, outType, outPolarity)

Purpose

Specifies the counting configuration to use for a counter.

Parameters

Input

Name	Type		Description
	Windows	Windows NT	
deviceNumber	I16	I32	assigned by configuration utility
ctr	I16	I32	counter number
edgeMode	I16	I32	count rising or falling edges
gateMode	I16	I32	gating mode to be used
outType	I16	I32	type of output generated
outPolarity	I16	I32	output polarity

Parameter Discussion

ctr is the counter number.

Range:	1, 2, or 5 for an MIO device except the E Series devices.
	1 through 10 for a PC-TIO-10.
	2 for an EISA-A2000.

edgeMode indicates which edge of the input signal that the counter should count. **edgeMode** must be either 0 or 1.

0:	counter counts rising edges.
1:	counter counts falling edges.

CTR_Config

Continued

gateMode selects the gating mode to be used by the counter. There are eight different gating modes. Each gating mode has been assigned a number between zero and 7. The available gating modes are as follows:

0:	No gating used.
1:	High-level gating of counter **ctr** used.
2:	Low-level gating of counter **ctr** used.
3:	Edge-triggered gating used—rising edge of counter **ctr**.
4:	Edge-triggered gating used—falling edge of counter **ctr**.
5:	Active high on terminal count of next lower-order counter.
6:	Active high on gate of next higher-order counter.
7:	Active high on gate of next lower-order counter.
8:	Special gating.

outType selects which type of output is to be generated by the counter. The counters generate two types of output signals: TC toggled output and TC pulse output.

0:	TC toggled output type used.
1:	TC pulse output type used.

outPolarity selects the output polarity used by the counter.

0:	Positive logic output.
1:	Negative logic (inverted) output.

Using This Function

If you select TC pulse output type, then **outPolarity** = 0 means that NI-DAQ generates active logic-high terminal count pulses. **outPolarity** = 1 means that NI-DAQ generates active logic-low terminal count pulses. Similarly, if you select TC toggled output type, then **outPolarity** = 0 means the OUT signal toggles from low to high on the first TC. **outPolarity** = 1 means the OUT signal toggles from high to low on the first TC.

`CTR_Config` saves the parameters in the configuration table for the specified counter. NI-DAQ uses this configuration table when the counter is set up for an event-counting, pulse output, or frequency output operation. You can use `CTR_Config` to take advantage of the many counter modes.

The default settings for the counter configuration modes are as follows:
edgeMode = 0: Counter counts rising edges.
gateMode = 0: No gating used.
outType = 0: TC toggled output type used.
outPolarity = 0: Positive logic output used.

CTR_Config

Continued

If you want to change the counter configuration from this default setting, you must call `CTR_Config` and indicate which configuration you want before initiating any other counter operation.

Counter configuration settings applied through this function persist when waveform generation functions use the same counter. For example, to externally trigger a waveform generation option, use this function to change the **gatemode** to 1 (high-level gating), and then call the waveform generation functions. The waveform generation will be delayed until a high-level signal appears on the gate pin on the I/O connector. Note that this is really not a trigger signal but is a gating signal, since the waveform generation will pause if the gate goes low at any time. Due to limitations of the Am9513 counter/timer chip, it is not possible to use **gateModes** 3 and 4. You are responsible for producing a signal that stays high for the duration of the waveform generation operation.

CTR_EvCount

Format

status = CTR_EvCount (deviceNumber, ctr, timebase, cont)

Purpose

Configures the specified counter for an event-counting operation and starts the counter.

Parameters

Input

Name	Type		Description
	Windows	Windows NT	
deviceNumber	I16	I32	assigned by configuration utility
ctr	I16	I32	counter number
timebase	I16	I32	timebase value
cont	I16	I32	whether counting continues

Parameter Discussion

ctr is the counter number.

Range:	1, 2, or 5 for an MIO device except the E Series devices.
	1 through 10 for a PC-TIO-10.
	2 for an EISA-A2000.

timebase selects the timebase, or resolution, to be used by the counter. **timebase** has the following possible values:

-1:	Internal 5 MHz clock used as timebase (200 ns resolution) (AT-MIO-16F-5, AT-MIO-64F-5, AT-MIO-16X, and PC-TIO-10 only).
0:	TC signal of **ctr**-1 used as timebase.
1:	Internal 1 MHz clock used as timebase (1 µs resolution).
2:	Internal 100 kHz clock used as timebase (10 µs resolution).
3:	Internal 10 kHz clock used as timebase (100 µs resolution).
4:	Internal 1 kHz clock used as timebase (1 ms resolution).

CTR_EvCount

Continued

5:	Internal 100 Hz clock used as timebase (10 ms resolution).
6:	SOURCE1 used as timebase if 1 ≤ **ctr** ≤ 5 or SOURCE 6 used as timebase if 6 ≤ **ctr** ≤ 10.
7:	SOURCE2 used as timebase if 1 ≤ **ctr** ≤ 5 or SOURCE 7 used as timebase if 6 ≤ **ctr** ≤ 10.
8:	SOURCE3 used as timebase if 1 ≤ **ctr** ≤ 5 or SOURCE 8 used as timebase if 6 ≤ **ctr** ≤ 10.
9:	SOURCE4 used as timebase if 1 ≤ **ctr** ≤ 5 or SOURCE 9 used as timebase if 6 ≤ **ctr** ≤ 10.
10:	SOURCE5 used as timebase if 1 ≤ **ctr** ≤ 5 or SOURCE 10 used as timebase if 6 ≤ **ctr** ≤ 10.
11:	GATE 1 used as timebase if 1 ≤ **ctr** ≤ 5 or GATE6 used as timebase if 6 ≤ **ctr** ≤ 10.
12:	GATE 2 used as timebase if 1 ≤ **ctr** ≤ 5 or GATE7 used as timebase if 6 ≤ **ctr** ≤ 10.
13:	GATE 3 used as timebase if 1 ≤ **ctr** ≤ 5 or GATE8 used as timebase if 6 ≤ **ctr** ≤ 10.
14:	GATE 4 used as timebase if 1 ≤ **ctr** ≤ 5 or GATE9 used as timebase if 6 ≤ **ctr** ≤ 10.
15:	GATE 5 used as timebase if 1 ≤ **ctr** ≤ 5 or GATE10 used as timebase if 6 ≤ **ctr** ≤ 10.

Set **timebase** to zero if you plan to concatenate counters. Set **timebase** to 1 through 5 for the counter to count one of the five available internal signals. Set **timebase** to 6 through 15 (except 10 for the PC-TIO-10) if you plan to provide an external signal to a counter. This external signal is then the signal NI-DAQ will count for event counting.

cont indicates whether counting continues after the counter reaches 65,535 and rolls over to zero. **cont** can be either zero or 1. If **cont** = 0, event counting stops when the counter reaches 65,535 and rolls over, in which case an overflow condition is registered. If **cont** = 1, event counting continues when the counter rolls over and no overflow condition is registered. **cont** = 1 is useful when more than one counter is concatenated for event counting.

Using This Function

`CTR_EvCount` configures the specified counter for an event-counting operation. The function configures the counter to count up from zero and to use the gating mode, edge mode, output type, and polarity as specified by the `CTR_Config` call.

CTR_EvCount

Continued

☞ **Note:** ***Edge gating mode does not operate properly during event counting if*** **cont** ***= 1. If*** **cont** ***= 1, use level gating modes or no-gating mode.***

Applications for `CTR_EvCount` are discussed in *Event-Counting Applications* in Chapter 3, *Software Overview*, of the *NI-DAQ User Manual for PC Compatibles*.

CTR_EvRead

Format

status = CTR_EvRead (deviceNumber, ctr, overflow, count)

Purpose

Reads the current counter total without disturbing the counting process and returns the count and overflow conditions.

Parameters

Input

Name	Type		Description
	Windows	Windows NT	
deviceNumber	I16	I32	assigned by configuration utility
ctr	I16	I32	counter number

Output

Name	Type		Description
	Windows	Windows NT	
overflow	I16	I32	overflow state of the counter
count	U16	U32	current total of the specified counter

CTR_EvRead

Continued

Parameter Discussion

ctr is the counter number.

Range: 1, 2, or 5 for an MIO device except the E Series devices.
1 through 10 for a PC-TIO-10.
2 for an EISA-A2000.

overflow returns the overflow state of the counter. A counter overflows if it counts up to 65,535 and rolls over to zero on the next count. If **overflow** = 0, no overflow has occurred. If **overflow** = 1, an overflow occurred. See the *Special Considerations for Overflow Detection* section later in this function.

count returns the current total of the specified counter. **count** can be between zero and 65,535. **count** represents the number of edges (either falling or rising edges, not both) that have occurred since the counter started counting.

☞ **Note:** ***C Programmers*—overflow *and* count *are pass-by-reference parameters.***

Using This Function

`CTR_EvRead` reads the current value of the counter without disturbing the counting process and returns the value in **count**. `CTR_EvRead` also performs overflow detection and returns the overflow status in **overflow**. Overflow detection and the significance of **count** depend on the counter configuration.

Special Considerations for Overflow Detection

For NI-DAQ to detect an overflow condition, you must configure the counter for TC toggled output type and positive output polarity, and then you must configure the counter to stop counting on overflow (**cont** = 0 in the `CTR_EvCount` call). If these conditions are not met, the value of **overflow** is meaningless. If more than one counter is concatenated for event-counting applications, you should configure the lower-order counters to continue counting when overflow occurs, and overflow detection is only meaningful for the highest order counter. **count**, returned by `CTR_EvRead` for the lower-order counters, then represents the module 65,536 event count. See *Event-Counting Applications* in Chapter 3, *Software Overview*, in the *NI-DAQ User Manual for PC Compatibles* for more information.

CTR_FOUT_Config

Format

status = CTR_FOUT_Config (deviceNumber, FOUT_port, mode, timebase, division)

Purpose

Disables or enables and sets the frequency of the 4-bit programmable frequency output.

Parameters

Input

Name	Type		Description
	Windows	Windows NT	
deviceNumber	I16	I32	assigned by configuration utility
FOUT_port	I16	I32	frequency output port
mode	I16	I32	enable or disable the programmable frequency output
timebase	I16	I32	timebase value
division	I16	I32	divide-down factor for generating the clock

Parameter Discussion

FOUT_port is the frequency output port to be programmed.

1: For FOUT1 on the PC-TIO-10 or FOUT on the MIO device, except the E Series devices.
2: For FOUT2 on the PC-TIO-10.

mode selects whether to enable or disable the programmable frequency output. **mode** can be 0 or 1.

0: The frequency output signal is turned off to a low-logic state.
1: The frequency output signal is enabled.

CTR_FOUT_Config

Continued

If **mode** = 0, none of the following parameters apply.

timebase selects the timebase, or resolution, to be used by the programmable frequency output. **timebase** has the following possible values:

-1:	Internal 5 MHz clock used as timebase (200 ns resolution) (AT-MIO-16F-5, AT-MIO-64F-5, AT-MIO-16X, and PC-TIO-10 only).
1:	Internal 1 MHz clock used as timebase (1 µs resolution).
2:	Internal 100 kHz clock used as timebase (10 µs resolution).
3:	Internal 10 kHz clock used as timebase (100 µs resolution).
4:	Internal 1 kHz clock used as timebase (1 ms resolution).
5:	Internal 100 Hz clock used as timebase (10 ms resolution).
6:	SOURCE1 used as timebase if **FOUT_port** = 1 or SOURCE 6 used as timebase if **FOUT_port** = 2.
7:	SOURCE2 used as timebase if **FOUT_port** = 1 or SOURCE 7 used as timebase if **FOUT_port** = 2.
8:	SOURCE3 used as timebase if **FOUT_port** = 1 or SOURCE 8 used as timebase if **FOUT_port** = 2.
9:	SOURCE4 used as timebase if **FOUT_port** = 1 or SOURCE 9 used as timebase if **FOUT_port** = 2.
10:	SOURCE5 used as timebase if **FOUT_port** = 1 or SOURCE 10 used as timebase if **FOUT_port** = 2.
11:	GATE 1 used as timebase if **FOUT_port** = 1 or GATE6 used as timebase if **FOUT_port** = 2.
12:	GATE 2 used as timebase if **FOUT_port** = 1 or GATE7 used as timebase if **FOUT_port** = 2.
13:	GATE 3 used as timebase if **FOUT_port** = 1 or GATE8 used as timebase if **FOUT_port** = 2.
14:	GATE 4 used as timebase if **FOUT_port** = 1 or GATE9 used as timebase if **FOUT_port** = 2.
15:	GATE 5 used as timebase if **FOUT_port** = 1 or GATE10 used as timebase if **FOUT_port** = 2.

division is the divide-down factor for generating the clock. The clock frequency is then equal to (**timebase** frequency)/**division**.
Range: 1 through 16.

Using This Function

Generates a 50% duty-cycle output clock at the programmable frequency output signal FOUT if **mode** = 1; otherwise, the FOUT signal is a low-logic state. The frequency of the FOUT signal is the timebase frequency divided by the division factor.

CTR_Period

Format

status = CTR_Period (deviceNumber, ctr, timebase)

Purpose

Configures the specified counter for period or pulse-width measurement.

Parameters

Input

Name	Type		Description
	Windows	Windows NT	
deviceNumber	I16	I32	assigned by configuration utility
ctr	I16	I32	counter number
timebase	I16	I32	timebase value

Parameter Discussion

ctr is the counter number.

Range: 1, 2, or 5 for an MIO device except the E Series devices.
1 through 10 for a PC-TIO-10.
2 for an EISA-A2000.

timebase selects the timebase, or resolution, to be used by the counter. **timebase** has the following possible values:

-1:	Internal 5 MHz clock used as timebase (200 ns resolution) (AT-MIO-16F-5, AT-MIO-64F-5, AT-MIO-16X, and PC-TIO-10 only).
0:	TC signal of **ctr**-1 used as timebase.
1:	Internal 1 MHz clock used as timebase (1 µs resolution).
2:	Internal 100 kHz clock used as timebase (10 µs resolution).
3:	Internal 10 kHz clock used as timebase (100 µs resolution).
4:	Internal 1 kHz clock used as timebase (1 ms resolution).
5:	Internal 100 Hz clock used as timebase (10 ms resolution).

CTR_Period

Continued

6: SOURCE1 used as timebase if 1 ≤ **ctr** ≤ 5 or SOURCE 6 used as timebase if 6 ≤ **ctr** ≤ 10.

7: SOURCE2 used as timebase if 1 ≤ **ctr** ≤ 5 or SOURCE 7 used as timebase if 6 ≤ **ctr** ≤ 10.

8: SOURCE3 used as timebase if 1 ≤ **ctr** ≤ 5 or SOURCE 8 used as timebase if 6 ≤ **ctr** ≤ 10.

9: SOURCE4 used as timebase if 1 ≤ **ctr** ≤ 5 or SOURCE 9 used as timebase if 6 ≤ **ctr** ≤ 10.

10: SOURCE5 used as timebase if 1 ≤ **ctr** ≤ 5 or SOURCE 10 used as timebase if 6 ≤ **ctr** ≤ 10.

11: GATE 1 used as timebase if 1 ≤ **ctr** ≤ 5 or GATE6 used as timebase if 6 ≤ **ctr** ≤ 10.

12: GATE 2 used as timebase if 1 ≤ **ctr** ≤ 5 or GATE7 used as timebase if 6 ≤ **ctr** ≤ 10.

13: GATE 3 used as timebase if 1 ≤ **ctr** ≤ 5 or GATE8 used as timebase if 6 ≤ **ctr** ≤ 10.

14: GATE 4 used as timebase if 1 ≤ **ctr** ≤ 5 or GATE9 used as timebase if 6 ≤ **ctr** ≤ 10.

15: GATE 5 used as timebase if 1 ≤ **ctr** ≤ 5 or GATE10 used as timebase if 6 ≤ **ctr** ≤ 10.

Set **timebase** to 0 if you plan to concatenate counters. Set **timebase** to 1 through 5 for the counter to count one of the five available internal signals. Set **timebase** to 6 through 15 (except 10 for the PC-TIO-10) if you plan to provide an external signal to a counter. This external signal is then the signal NI-DAQ will count for event counting.

Using This Function

`CTR_Period` configures the specified counter for period and pulse-width measurement. The function configures the counter to count up from zero and to use the gating mode, edge mode, output type, and polarity as specified by the `CTR_Config` call.

Applications for `CTR_Period` are discussed in the section *Period and Continuous Pulse-Width Measurement Applications* in Chapter 3, *Software Overview,* of the *NI-DAQ User Manual for PC Compatibles*.

CTR_Pulse

Format

status = CTR_Pulse (deviceNumber, ctr, timebase, delay, pulseWidth)

Purpose

Causes the specified counter to generate a specified pulse-programmable delay and pulse width.

Parameters

Input

Name	Type		Description
	Windows	Windows NT	
deviceNumber	I16	I32	assigned by configuration utility
ctr	I16	I32	counter number
timebase	I16	I32	timebase value
delay	U16	U32	interval before the pulse
pulseWidth	U16	U32	interval of the pulse

Parameter Discussion

ctr is the counter number.

Range: 1, 2, or 5 for an MIO device except the E Series devices.
1 through 10 for a PC-TIO-10.
2 for an EISA-A2000.

timebase selects the timebase, or resolution, to be used by the counter. **timebase** has the following possible values:

-1: Internal 5 MHz clock used as timebase (200 ns resolution) (AT-MIO-16F-5, AT-MIO-64F-5, AT-MIO-16X, and PC-TIO-10 only).
0: TC signal of **ctr**-1 used as timebase.
1: Internal 1 MHz clock used as timebase (1 µs resolution).
2: Internal 100 kHz clock used as timebase (10 µs resolution).

CTR_Pulse

Continued

3:	Internal 10 kHz clock used as timebase (100 µs resolution).
4:	Internal 1 kHz clock used as timebase (1 ms resolution).
5:	Internal 100 Hz clock used as timebase (10 ms resolution).
6:	SOURCE1 used as timebase if 1 ≤ **ctr** ≤ 5 or SOURCE 6 used as timebase if 6 ≤ **ctr** ≤ 10.
7:	SOURCE2 used as timebase if 1 ≤ **ctr** ≤ 5 or SOURCE 7 used as timebase if 6 ≤ **ctr** ≤ 10.
8:	SOURCE3 used as timebase if 1 ≤ **ctr** ≤ 5 or SOURCE 8 used as timebase if 6 ≤ **ctr** ≤ 10.
9:	SOURCE4 used as timebase if 1 ≤ **ctr** ≤ 5 or SOURCE 9 used as timebase if 6 ≤ **ctr** ≤ 10.
10:	SOURCE5 used as timebase if 1 ≤ **ctr** ≤ 5 or SOURCE 10 used as timebase if 6 ≤ **ctr** ≤ 10.
11:	GATE 1 used as timebase if 1 ≤ **ctr** ≤ 5 or GATE6 used as timebase if 6 ≤ **ctr** ≤ 10.
12:	GATE 2 used as timebase if 1 ≤ **ctr** ≤ 5 or GATE7 used as timebase if 6 ≤ **ctr** ≤ 10.
13:	GATE 3 used as timebase if 1 ≤ **ctr** ≤ 5 or GATE8 used as timebase if 6 ≤ **ctr** ≤ 10.
14:	GATE 4 used as timebase if 1 ≤ **ctr** ≤ 5 or GATE9 used as timebase if 6 ≤ **ctr** ≤ 10.
15:	GATE 5 used as timebase if 1 ≤ **ctr** ≤ 5 or GATE10 used as timebase if 6 ≤ **ctr** ≤ 10.

Set **timebase** to 0 if you plan to concatenate counters. Set **timebase** to 1 through 5 for the counter to use one of the five available internal signals. Set **timebase** to 6 through 15 (except 10 for the PC-TIO-10) if you plan to provide an external clock to the counter.

delay is the delay before NI-DAQ generates the pulse. **delay** can be between 3 and 65,535. Use the following formula to determine the actual time period that **delay** represents:

delay ∗ (**timebase** resolution)

pulseWidth is the width of the pulse NI-DAQ will generate. **pulseWidth** can be between 0 and 65,535. Use the following formula to determine the actual time that **pulseWidth** represents:

pulseWidth ∗ (**timebase** resolution)

CTR_Pulse

Continued

for 1 ≤ **pulseWidth** ≤ 65,535. **pulseWidth** = 0 is a special case of pulse generation and actually generates a pulse of infinite duration (see the timing diagrams in Figures 2-9 and 2-10).

Using This Function

`CTR_Pulse` sets up the counter to generate a pulse of the duration specified by **pulseWidth**, after a time delay of the duration specified by **delay**. If you specify no gating, `CTR_Pulse` starts the counter; otherwise, counter operation is controlled by the gate input. The selected timebase determines the timing of pulse generation as shown in Figure 2-9.

You can generate successive pulses by calling `CTR_Restart` or `CTR_Pulse` again. Be sure that the delay period of the previous pulse has elapsed before calling `CTR_Restart` or `CTR_Pulse`. A successive call will wait until the previous pulse is completed before generating the next pulse. In the case where **pulseWidth** = 0 and TC toggle output is used, the output polarity toggles after every call to `CTR_Restart`.

Pulse Generation Timing Considerations

Figure 2-9 shows pulse generation timing for both the TC toggled output and TC pulse output cases. These signals are positive polarity output signals.

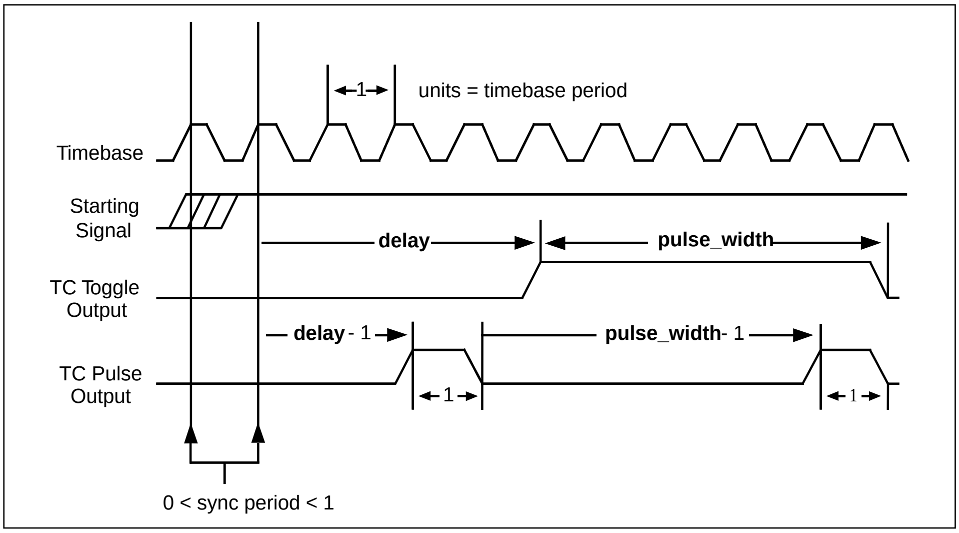


Figure 2-9. Pulse Generation Timing

CTR_Pulse

Continued

An uncertainty is associated with the delay period due to counter synchronization. Counting starts on the first timebase edge *after* NI-DAQ applies the starting signal. The time between receipt of the starting pulse and start of pulse generation can be between (**delay**) and (**delay**+1) units of the timebase in duration.

pulseWidth = 0 generates a special case signal as shown in Figure 2-10.

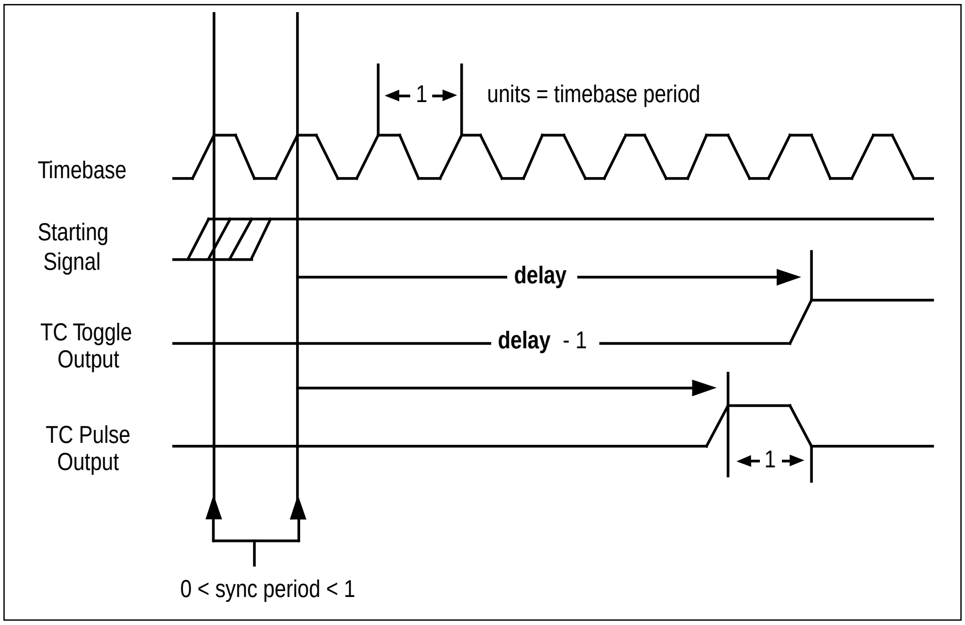


Figure 2-10. Pulse Timing for pulseWidth = 0

CTR_Rate

Format

status = CTR_Rate (freq, duty, timebase, period1, period2)

Purpose

Converts frequency and duty-cycle values of a square wave you want into the timebase and period parameters needed for input to the `CTR_Square` function that produces the square wave.

Parameters

Input

Name	**Type**		**Description**
	Windows	**Windows NT**	
freq	F64	F64	frequency you want
duty	F64	F64	duty cycle you want

Output

Name	**Type**		**Description**
	Windows	**Windows NT**	
timebase	I16	I32	onboard source signal used
period1	U16	U32	units of time that the square wave is high
period2	U16	U32	units of time that the square wave is low

CTR_Rate

Continued

Parameter Discussion

freq is the square wave frequency you want in cycles per second (Hz).
Range: 0.0008 through 2,500,000 Hz.

duty is the square wave duty cycle you want as a fraction. With positive output polarity and TC toggled output selected, the fraction expressed by **duty** describes the fraction of a single wavelength of the square wave that is logical high.
Range: 0.0 through 1.0 noninclusive (that is, any value between, but not including, 0.0 and 1.0).

timebase is a code that represents the resolution of the onboard source signal that the counter uses to produce the square wave. You can input the value returned by **timebase** directly to the `CTR_Square` function.

1:	1 µs.
2:	10 µs.
3:	100 µs.
4:	1 ms.
5:	10 ms.

period1 and **period2** represent the number of units of time (selected by **timebase**) that the square wave is high and low, respectively. The roles of **period1** and **period2** are reversed if the output polarity is negative.
Range: 1 through 65,535.

☞ **Note:** ***C Programmers*—timebase, period1,** ***and*** **period2** ***are pass-by-reference parameters.***

Using This Function

`CTR_Rate` translates a definition of a square wave in terms of frequency and duty cycle into terms of a timebase and two period values. You can then directly input the timebase and period values into the `CTR_Square` function to produce the square wave you want.

`CTR_Rate` emphasizes matching the frequency first and then the duty cycle. That is, if the duty fraction is 0.5 but an odd-numbered total period is needed to produce the frequency you want, the two periods returned by `CTR_Rate` will not be equal and the duty cycle of the square wave will differ slightly from 50 percent. For example, if **freq** is 40,000 Hz and **duty** is 0.50, `CTR_Rate` returns values of 1 for **timebase**, 13 for **period1**, and 12 for **period2**. The resulting square wave has the frequency of 40,000 Hz but a duty fraction of 0.52.

CTR_Reset

Format

status = CTR_Reset (deviceNumber, ctr, output)

Purpose

Turns off the specified counter operation and places the counter output drivers in the selected output state.

Parameters

Name	Type		Description
	Windows	Windows NT	
deviceNumber	I16	I32	assigned by configuration utility
ctr	I16	I32	counter number
output	I16	I32	output state of the counter OUT signal driver

Parameter Discussion

ctr is the counter number.

Range:	1, 2, or 5 for an MIO device except the E Series devices.
	1 through 10 for a PC-TIO-10.
	2 for an EISA-A2000.

output indicates the output state of the counter OUT signal driver. **output** can be between 0 and 2 and represents three choices of output state.

0:	Set OUT signal driver to high-impedance state.
1:	Set OUT signal driver to low-logic state.
2:	Set OUT signal driver to high-logic state.

Using This Function

`CTR_Reset` causes the specified counter to terminate its current operation, clears the counter mode, and places the counter OUT driver in the specified output state. Once a counter has performed an operation (a square wave, for example), you must use

CTR_Reset

Continued

`CTR_Reset` to stop and clear the counter before setting it up for any subsequent operation of a different type (event counting, for example). You can also use `CTR_Reset` to change the output state of an idle counter.

☞ **Note:** ***The output line of counter 1 on the MIO-16 and AT-MIO-16D, and counters 1, 2, and 5 on the AT-MIO-16F-5, AT-MIO-64F-5, and AT-MIO-16X are pulled up to +5 V while in the high-impedance state.***

CTR_Restart

Format

status = CTR_Restart (deviceNumber, ctr)

Purpose

Restarts operation of the specified counter.

Parameters

Input

Name	Type		Description
	Windows	**Windows NT**	
deviceNumber	I16	I32	assigned by configuration utility
ctr	I16	I32	counter number

Parameter Discussion

ctr is the counter number.

Range:	1, 2, or 5 for an MIO device except the E Series devices.
	1 through 10 for a PC-TIO-10.
	2 for an EISA-A2000.

Using This Function

You can use `CTR_Restart` after a `CTR_Stop` operation to allow the suspended counter to resume. If the specified counter was never set up for an operation, `CTR_Restart` returns an error.

You can also use `CTR_Restart` after a `CTR_Pulse` operation to generate additional pulses. `CTR_Pulse` generates the first pulse. In this case, do not call `CTR_Restart` until after the previous pulse has completed.

CTR_Simul_Op

Format

status = CTR_Simul_Op (deviceNumber, numCtrs, ctrList, mode)

Purpose

Configures and simultaneously starts and stops multiple counters.

Parameters

Input

Name	Type		Description
	Windows	Windows NT	
deviceNumber	I16	I32	assigned by configuration utility
numCtrs	I16	I32	number of counters to operate
ctrList	[I16]	[I32]	array of counter numbers
mode	I16	I32	operating mode

Parameter Discussion

numCtrs is the number of counters to which the operation is performed.
Range: 1 through 10.

ctrList is an array of integers of size **numCtrs** containing the counter numbers of the counters for performing the operation.
Range: 1, 2, or 5 for an MIO device except the E Series devices.
1 through 10 for a PC-TIO-10.
2 for an EISA-A2000.

☞ **Note:** ***By default, counters are not reserved for simultaneous operations.***

mode is the operating mode to be performed by this call.

0: Cancel reservation of counters specified in **ctrList**.

1: Reserve counters specified in **ctrList** for simultaneous start, restart, stop, or count latch operation.

CTR_Simul_Op

Continued

2: Perform a simultaneous start/restart on the counters specified in **ctrList**.
3: Perform a simultaneous stop on the counters specified in **ctrList**.
4: Perform a simultaneous count latch on the counters specified in **ctrlist**. The counters must have been started by a previous call to `CTR_EvCount`. The counts can be retrieved one at a time by subsequent calls to `CTR_EvRead`.

☞ **Note:** ***It is not necessary to call*** `CTR_Simul_Op` ***with mode set to 1 before calling*** `CTR_Simul_Op` ***with*** **mode** ***set to 4. That is, it is permissible to start two or more counters at different times and still latch their counts at the same time.***

Using This Function

You can simultaneously start multiple counters for any combination of event counting, square wave generation, or pulse generation. The following sequence is an example of using `CTR_Simul_Op`:

1. Specify the counters to use by putting their counter numbers into the **ctrList** array.
2. Call `CTR_Simul_Op` with **mode** = 1 to reserve these counters.
3. Set up the counters by calling `CTR_EvCount`, `CTR_Period`, `CTR_Square`, or `CTR_Pulse` for each reserved counter. Because these counters are reserved, they will not start immediately by those calls.
4. Call `CTR_Simul_Op` with **mode** = 2 to start these counters.
5. Call `CTR_Simul_Op` with **mode** = 3 to stop these counters.
6. Call `CTR_Simul_Op` with **mode** = 0 to free counters for non-simultaneous operations.

You can simultaneously stop counters from performing `CTR_EvCount`, `CTR_Period`, `CTR_Square`, or `CTR_Pulse` whether they were started by `CTR_Simul_Op` or not.

Trying to simultaneously start unreserved counters causes this function to return an error.

Call `CTR_Simul_Op` with **mode** = 0 to cancel the reserved status of counters specified in **ctrList**.

☞ **Note:** ***On the PC-TIO-10, the 10 counters are included on two counter/timer chips. These counter/timer chips are programmed sequentially. Simultaneous start-and-stop operations that specify counters from both***

CTR_Simul_Op

Continued

chips will experience a delay between the counters on the first chip (counters 1 through 5) and those on the second chip (counters 6 through 10). NI-DAQ returns a warning condition.

CTR_Square

Format

status = CTR_Square (deviceNumber, ctr, timebase, period1, period2)

Purpose

Causes the specified counter to generate a continuous square wave output of specified duty cycle and frequency.

Parameters

Input

Name	**Type**		**Description**
	Windows	**Windows NT**	
deviceNumber	I16	I32	assigned by configuration utility
ctr	I16	I32	counter number
timebase	I16	I32	timebase value
period1	U16	U32	period of the square wave
period2	U16	U32	period of the square wave

Parameter Discussion

ctr is the counter number.

Range:	1, 2, or 5 for an MIO device except the E Series devices.
	1 through 10 for a PC-TIO-10.
	2 for an EISA-A2000.

CTR_Square

Continued

timebase is the timebase, or resolution, to be used by the counter. **timebase** has the following possible values:

-1:	Internal 5 MHz clock used as timebase (200 ns resolution) (AT-MIO-16F-5, AT-MIO-64F-5, AT-MIO-16X, and PC-TIO-10 only).
0:	TC signal of **ctr**-1 used as timebase.
1:	Internal 1 MHz clock used as timebase (1 µs resolution).
2:	Internal 100 kHz clock used as timebase (10 µs resolution).
3:	Internal 10 kHz clock used as timebase (100 µs resolution).
4:	Internal 1 kHz clock used as timebase (1 ms resolution).
5:	Internal 100 Hz clock used as timebase (10 ms resolution).
6:	SOURCE1 used as timebase if 1 ≤ **ctr** ≤ 5 or SOURCE 6 used as timebase if 6 ≤ **ctr** ≤ 10.
7:	SOURCE2 used as timebase if 1 ≤ **ctr** ≤ 5 or SOURCE 7 used as timebase if 6 ≤ **ctr** ≤ 10.
8:	SOURCE3 used as timebase if 1 ≤ **ctr** ≤ 5 or SOURCE 8 used as timebase if 6 ≤ **ctr** ≤ 10.
9:	SOURCE4 used as timebase if 1 ≤ **ctr** ≤ 5 or SOURCE 9 used as timebase if 6 ≤ **ctr** ≤ 10.
10:	SOURCE5 used as timebase if 1 ≤ **ctr** ≤ 5 or SOURCE 10 used as timebase if 6 ≤ **ctr** ≤ 10.
11:	GATE 1 used as timebase if 1 ≤ **ctr** ≤ 5 or GATE6 used as timebase if 6 ≤ **ctr** ≤ 10.
12:	GATE 2 used as timebase if 1 ≤ **ctr** ≤ 5 or GATE7 used as timebase if 6 ≤ **ctr** ≤ 10.
13:	GATE 3 used as timebase if 1 ≤ **ctr** ≤ 5 or GATE8 used as timebase if 6 ≤ **ctr** ≤ 10.
14:	GATE 4 used as timebase if 1 ≤ **ctr** ≤ 5 or GATE9 used as timebase if 6 ≤ **ctr** ≤ 10.
15:	GATE 5 used as timebase if 1 ≤ **ctr** ≤ 5 or GATE10 used as timebase if 6 ≤ **ctr** ≤ 10.

Set **timebase** to 0 if you plan to concatenate counters. Set **timebase** to 1 through 5 for the counter to use one of the five available internal signals. Set **timebase** to 6 through 15 (except 10 for the PC-TIO-10) if you plan to provide an external clock to the counter.

CTR_Square

Continued

period1 and **period2** specify the two periods making up the square wave to be generated. For TC toggled output type and positive output polarity, **period1** indicates the duration of the on-cycle (high-logic state) and **period2** indicates the duration of the off-cycle (low-logic state).
Range: 1 through 65,535.

Using This Function

`CTR_Square` sets up the counter to generate a square wave of duration and frequency determined by **period1**, **period2**, and **timebase**. If you specify no gating, the function initiates square wave generation; otherwise, counter operation is controlled by the gate input.

The total period of the square wave is determined by the following formula:

$$(\textbf{period1} + \textbf{period2}) * (\textbf{timebase}\ \text{period})$$

This implies that the frequency of the square wave is as follows:

$$\frac{1}{(\textbf{period1} + \textbf{period2}) * (\textbf{timebase}\ \text{period})}$$

The percent duty cycle of the square wave is determined by the following formula:

$$\frac{\textbf{period 1}}{(\textbf{period1} + \textbf{period2}) * 100\%}$$

Figure 2-11 shows the timing of square wave generation for both TC toggled output and TC pulse output. For this example, **period1** = 3 and **period2** = 2. The output signals shown are positive polarity output signals.

When you use special gating (**gateMode** = 8), you can achieve gate-controlled pulse generation. When the gate input is high, NI-DAQ uses **period1** to generate the pulses. When the gate input is low, NI-DAQ uses **period2** to generate the pulses. If the output mode is TC Toggled, the result is two 50 percent duty square waves of difference

Continued

frequencies. If the output mode is TC Pulse, the result is two pulse trains of different frequencies.

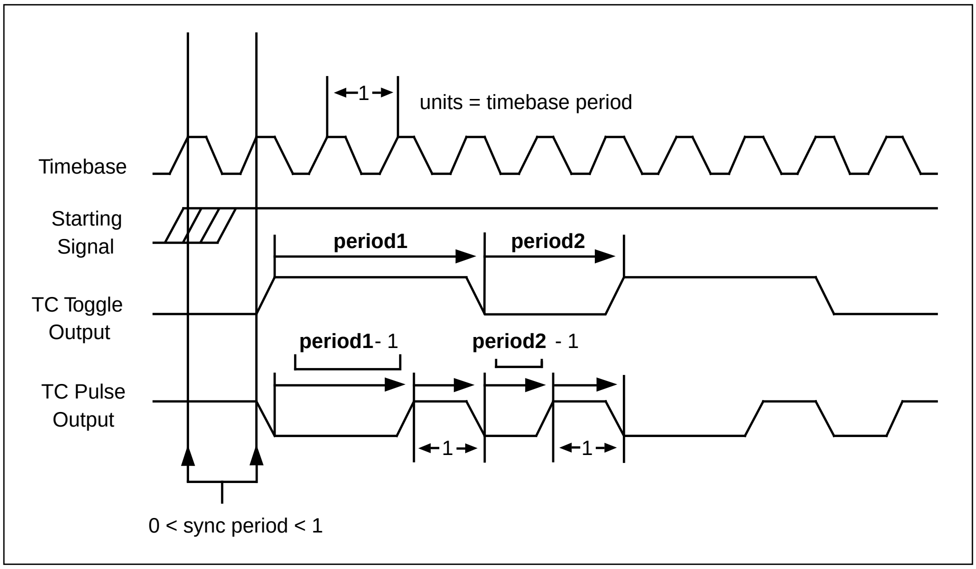


Figure 2-11. Square Wave Timing

Square Wave Generation Timing Considerations

There is an uncertainty associated with the beginning of square wave generation due to counter synchronization. Square wave generation starts on the first timebase edge *after* NI-DAQ applies the starting signal. The time between receipt of the starting signal and the start of the square wave generation can be between 0 and 1 units of the timebase in duration.

You should not use edge gating with square wave generation. If you use edge gating, the waveform stops after **period1** expires and then continues for one total period (**period2** + **period1**) only after NI-DAQ applies another edge. This behavior may or may not be useful. For continuous square wave generation, use level or no gating.

CTR_State

Format

status = CTR_State (deviceNumber, ctr, outState)

Purpose

Returns the OUT logic level of the specified counter.

Parameters

Input

Name	Type		Description
	Windows	Windows NT	
deviceNumber	I16	I32	assigned by configuration utility
ctr	I16	I32	counter number

Output

Name	Type		Description
	Windows	Windows NT	
outState	I16	I32	returns the logic level of the counter OUT signal

Parameter Discussion

ctr is the counter number.

Range:	1, 2, or 5 for an MIO device except the E Series devices.
	1 through 10 for a PC-TIO-10.
	2 for an EISA-A2000.

outState returns the logic level of the counter OUT signal. **outState** is either 0 or 1.

0:	Indicates that OUT is at a low-logic state.
1:	Indicates that OUT is at a high-logic state.

CTR_State

Continued

☞ **Note:** ***C Programmers*—outState *is a pass-by-reference parameter.***

Using This Function

`CTR_State` reads the logic state of the OUT signal of the specified counter and returns the state in **outState**. If the counter OUT driver is set to the high-impedance state, **outState** is indeterminate and can be either 0 or 1.

CTR_Stop

Format

status = CTR_Stop (deviceNumber, ctr)

Purpose

Suspends operation of the specified counter so that you can restart the counter operation.

Parameters

Input

Name	Type		Description
	Windows	Windows NT	
deviceNumber	I16	I32	assigned by configuration utility
ctr	I16	I32	counter number

Parameter Discussion

ctr is the counter number.

Range: 1, 2, or 5 for an MIO device except the E Series devices.
1 through 10 for a PC-TIO-10.
2 for an EISA-A2000.

Using This Function

`CTR_Stop` suspends the operation of the counter in such a way that the counter can be restarted by `CTR_Restart` and continue in its operation. For example, if a counter is set up for frequency output, issuing `CTR_Stop` causes the counter to stop generating a square wave, and `CTR_Restart` allows it to resume. `CTR_Stop` causes the counter output to remain at the state it was in when `CTR_Stop` was issued.

☞ **Note:** ***Due to hardware limitations,*** `CTR_Stop` ***cannot stop a counter generating a square wave with period1 of 1 and period2 of 1.***

DAQ_Check

Format

status = DAQ_Check (deviceNumber, daqStopped, retrieved)

Purpose

Checks whether the current DAQ operation is complete and returns the status and the number of samples acquired to that point.

Parameters

Input

Name	Type		Description
	Windows	Windows NT	
deviceNumber	I16	I32	assigned by configuration utility

Output

Name	Type		Description
	Windows	Windows NT	
daqStopped	I16	I32	indication of whether the data acquisition has completed
retrieved	U32	U32	progress of an acquisition

Parameter Discussion

daqStopped returns an indication of whether the data acquisition has completed.

0: The DAQ operation is not yet complete.
1: The DAQ operation has stopped. Either the buffer is full or an error has occurred.

retrieved indicates the progress of an acquisition. The meaning of **retrieved** depends on whether pretrigger mode has been enabled (see `DAQ_StopTrigger_Config`). If

DAQ_Check

Continued

pretrigger mode is disabled, **retrieved** returns the number of samples collected by the acquisition at the time of the call to `DAQ_Check`. The value of **retrieved** increases until it equals the **count** indicated in the call that initiated the acquisition, at which time the acquisition terminates. However, if pretrigger mode is enabled, **retrieved** returns the offset of the position in your buffer where the next data point is placed when it is acquired. Once the value of **retrieved** reaches **count**-1 and rolls over to 0, the acquisition begins to overwrite old data with new data. When NI-DAQ applies a signal to the stop trigger input, the acquisition collects an additional number of samples indicated by **ptsAfterStoptrig** in the call to `DAQ_StopTrigger_Config` and then terminates. When `DAQ_Check` returns a status of 1, **retrieved** contains the offset of the oldest data point in the array (assuming that the acquisition has written to the entire buffer at least once). In pretrigger mode, `DAQ_Check` automatically rearranges the array upon completion of the acquisition so that the oldest data point is at the beginning of the array. Thus, **retrieved** always equals 0 upon completion of a pretrigger mode acquisition.

☞ **Note:** ***C Programmers*—daqStopped *and* retrieved *are pass-by-reference parameters.***

Using This Function

`DAQ_Check` checks the current background DAQ operation to determine whether it has completed and returns the number of samples acquired at the time that you called `DAQ_Check`. If the operation is complete, `DAQ_Check` sets **daqStopped** = 1. Otherwise, **daqStopped** is set to 0. Before `DAQ_Check` returns **daqStopped** = 1, it calls `DAQ_Clear`, allowing another Start call to execute immediately.

If `DAQ_Check` returns an **overFlowError** or an **overRunError**, the DAQ operation is terminated; some A/D conversions were lost due to a sample rate that is too high (sample interval was too small). An **overFlowError** indicates that the A/D FIFO memory overflowed because the DAQ servicing operation was not able to keep up with sample rate. An **overRunError** indicates that the DAQ circuitry was not able to keep up with the sample rate. Before returning either of these error codes, `DAQ_Check` calls `DAQ_Clear` to terminate the operation and reinitialize the DAQ circuitry.

DAQ_Clear

Format

status = DAQ_Clear (deviceNumber)

Purpose

Cancels the current DAQ operation (both single-channel and multiple-channel scanned) and reinitializes the DAQ circuitry.

Parameters

Input

Name	Type		Description
	Windows	Windows NT	
deviceNumber	I16	I32	assigned by configuration utility

Using This Function

`DAQ_Clear` turns off any current DAQ operation (both single-channel and multiple-channel), cancels the background process that is handling the data acquisition, and clears any error flags set as a result of the data acquisition. NI-DAQ then reinitializes the DAQ circuitry so that NI-DAQ can start another data acquisition.

☞ **Note:** ***If your application calls*** `DAQ_Start`***,*** `SCAN_Start`***, or*** `Lab_ISCAN_Start`***, always make sure that you call*** `DAQ_Clear` ***before your application terminates and returns control to the operating system. Unpredictable behavior can result unless you make this call (either directly, or indirectly through*** `DAQ_Check`***,*** `Lab_ISCAN_Check`***, or*** `DAQ_DB_Transfer`***).***

DAQ_Config

Format

status = DAQ_Config (deviceNumber, startTrig, extConv)

Purpose

Stores configuration information for subsequent DAQ operations.

Parameters

Input

Name	Type		Description
	Windows	Windows NT	
deviceNumber	I16	I32	assigned by configuration utility
startTrig	I16	I32	whether the trigger to initiate data acquisition is generated externally
extConv	I16	I32	selects A/D conversion clock source

Parameter Discussion

startTrig indicates whether the trigger to initiate DAQ sequences is generated externally.

0: Generate software trigger to start DAQ sequence (the default).

1: Wait for external trigger pulse at STARTTRIG of the MIO-16 and AT-MIO-16D, or at EXTTRIG* of the AT-MIO-16F-5, AT-MIO-64F-5, and AT-MIO-16X, or at EXTTRIG of the Lab and 1200 Series devices to initiate DAQ sequence (not valid for 516 and LPM devices and the DAQCard-500/700).

extConv indicates whether the timing of A/D conversions during the DAQ sequence is controlled externally or internally with the sample-interval and/or scan-interval clocks.

0:	Use onboard clock(s) to control data acquisition sample-interval and scan-interval timing (the default).
1:	Allow external clock to control sample-interval timing.
2:	Allow external clock to control scan-interval timing (MIO, AI, and Lab and 1200 Series devices only).
3:	Allow external control of sample-interval timing and scan-interval timing (AT-MIO-16F-5, AT-MIO-64F-5, AT-MIO-16X, and Lab and 1200 Series devices only).

If you are using an E Series device, see the `Select_Signal` function for information about the external timing signals.

Using This Function

`DAQ_Config` saves the parameters in the configuration table for future data acquisition. `DAQ_Start` and `SCAN_Start` use the configuration table to set the DAQ circuitry to the correct timing modes.

If both **startTrig** and **extConv** are 0, A/D conversions begin as soon as you call `DAQ_Start`, `SCAN_Start`, or `Lab_ISCAN_Start`. When **startTrig** is 1, A/D conversions do not begin until NI-DAQ receives an external trigger pulse. In the latter case, the Start call merely arms the device. If you are using all E Series devices, see the `Select_Signal` function for information about the external timing signals. Once the A/D conversions have begun (with the start trigger), the onboard counters control the timing of the conversions. When **extConv** is 1, the timing of A/D conversions is completely controlled by the signal applied at the EXTCONV* input. Again, the Start call merely arms the device, and after you make this call, the device performs an A/D conversion every time NI-DAQ receives a pulse at the EXTCONV* input. When **extConv** is 2, the device performs a multiple-channel scan each time the device receives an active low pulse at the OUT2 signal (pin 46) of the MIO-16 I/O connector, or the COUTB1 signal (pin 43) on Lab and 1200 Series devices.

On the MIO-16 and AT-MIO-16D it is not possible to simultaneously use both external start triggering and external sample clock (**startTrig** = 1 and **extConv** = 1). NI-DAQ returns an error if you attempt to configure them simultaneously. On the AT-MIO-16F-5, AT-MIO-64F-5, AT-MIO-16X, and Lab and 1200 Series devices, you can configure external start triggering and the external sample clock simultaneously.

DAQ_Config

Continued

(MIO-16 and Lab and 1200 Series devices only) In most cases, you should not use external conversion pulses in scanning operations when you are using SCXI in Multiplexed mode. There is no way of masking conversions before the data acquisition begins, so any conversion pulses that occur before NI-DAQ triggers the acquisition will advance the SCXI channels. The AT-MIO-16X and AT-MIO-16F-5 do not have this restriction.

(Lab and 1200 Series devices only) If the device is using an external timing clock for A/D conversions (**extConv**= 1), the first clock pulse after one of the three start calls (`AI_Setup`, `DAQ_Start`, or `Lab_ISCAN_Start`) is to activate the device for external timing. It does not generate a conversion. However, all subsequent clock pulses will generate conversions.

(E Series devices only) If you use this function with **startTrig** = 1, the device waits for an active low external pulse on the PFI0 pin to initiate the DAQ sequence. If you use this function with **extConv** = 1 or 3, the device uses active low pulses on the PFI2 pin for sample-interval timing. If you use this function with **extConv** = 2 or 3, the device uses active low pulses on the PFI7 pin for scan-interval timing. These settings are consistent with the Am9513-based device selections. You can use the `Select_Signal` function instead of this function to take advantage of the DAQ-STC signal routing and polarity selection features.

The default settings for DAQ modes are as follows:
startTrig = 0: DAQ sequences are initiated through software.
extConv = 0: Onboard clock(s) are used to time A/D conversions.

If you want a DAQ timing configuration different from the default setting, you must call `DAQ_Config` with the configuration you want before initiating any DAQ sequences. You need to call `DAQ_Config` only when you change the DAQ configuration from the default setting.

If you want to scan channels on an SCXI-1140 module using an external Track*/Hold signal, you should call `DAQ_Config` with **extConv** = 2 so that the Track*/Hold signal of the module can control the scan interval timing during the acquisition.

The configuration information for the analog input circuitry is controlled by the `AI_Configure` call. This configuration information also affects data acquisition.

You cannot use pretrigger mode in conjunction with external conversion method on the MIO-16 and AT-MIO-16D.

DAQ_DB_Config

Format

status = DAQ_DB_Config (deviceNumber, DBmode)

Purpose

Enables or disables double-buffered DAQ operations.

Parameters

Input

Name	Type		Description
	Windows	Windows NT	
deviceNumber	I16	I32	assigned by configuration utility
DBmode	I16	I32	enable or disable double-buffered mode

Parameter Discussion

DBmode indicates whether to enable or disable the double-buffered mode of acquisition.

0: Disable double buffering (default).
1: Enable double buffering.

Using This Function

Double-buffered data acquisition cyclically fills a buffer with acquired data. The buffer is divided into two equal halves so that NI-DAQ can save data from one half while filling the other half. This mechanism makes it necessary to alternately save both halves of the buffer so that NI-DAQ does not overwrite data in the buffer before saving the data. Use the `DAQ_DB_Transfer` or `DAQ_DB_StrTransfer` functions to save the data as NI-DAQ acquires it. For additional information, see Chapter 5, *NI-DAQ Double Buffering*, of the *NI-DAQ User Manual for PC Compatibles*.

DAQ_DB_HalfReady

Format

status = DAQ_DB_HalfReady (deviceNumber, halfReady, daqStopped)

Purpose

Checks whether the next half buffer of data is available during a double-buffered data acquisition. You can use `DAQ_DB_HalfReady` to avoid the waiting period that can occur because the double-buffered transfer functions (`DAQ_DB_Transfer` and `DAQ_DB_StrTransfer`) wait until the data is ready before retrieving and returning it.

Parameters

Input

Name	Type		Description
	Windows	Windows NT	
deviceNumber	I16	I32	assigned by configuration utility

Output

Name	Type		Description
	Windows	Windows NT	
halfReady	I16	I32	whether the next half buffer of data is available
daqStopped	I16	I32	whether the data acquisition has completed

DAQ_DB_HalfReady

Continued

Parameter Discussion

halfReady indicates whether the next half buffer of data is available. When **halfReady** equals 1, you can use `DAQ_DB_Transfer` or `DAQ_DB_StrTransfer` to retrieve the data immediately. When **halfReady** equals 0, the data is not yet available.

daqstopped returns an indication of whether the data acquisition has completed. If **daqstopped** = 1, the DAQ operation is complete (or halted due to an error). If **daqstopped** = 0, the DAQ operation is still running.

☞ **Note:** ***C Programmers*—halfReady *and* daqstopped *are pass-by-reference parameters.***

Using This Function

Double-buffered data acquisition cyclically fills a buffer with acquired data. The buffer is divided into two equal halves so that NI-DAQ can save data from one half while filling the other half. This mechanism makes it necessary to alternately save both halves of the buffer so that NI-DAQ does not overwrite data in the buffer before saving the data. Use the `DAQ_DB_Transfer` or `DAQ_DB_StrTransfer` functions to save the data as NI-DAQ acquires it. Both of these functions, when called, wait for the data to become available before retrieving it and returning. During slower paced acquisitions this waiting period can be significant. You can use `DAQ_DB_HalfReady` to ensure that the transfer functions are called only when the data is already available.

DAQ_DB_StrTransfer

Format

status = DAQ_DB_StrTransfer (deviceNumber, strBuffer, ptsTfr, daqStopped)

Purpose

Transfers data from a circular buffer to a character buffer or a BASIC string during a double-buffered acquisition and waits until the data to be transferred is available before returning. `DAQ_DB_StrTransfer` is intended for BASIC applications using double-buffered data acquisition where data is saved on disk as it is acquired. You can then use the BASIC PUT statement to write the string to a disk file.

Parameters

Input

Name	Type		Description
	Windows	Windows NT	
deviceNumber	I16	I32	assigned by configuration utility

Output

Name	Type		Description
	Windows	Windows NT	
strBuffer	[U8]	[U8]	buffer to which the data is to be transferred
ptsTfr	U32	U32	number of points transferred
daqStopped	I16	I32	indicates the completion of a pretrigger mode acquisition

DAQ_DB_StrTransfer

Continued

Parameter Discussion

strBuffer is the buffer to which the data is to be transferred. The size of the buffer must be at least half the size of the large buffer being used for double-buffered data acquisition.

ptsTfr is the number of points transferred to **strBuffer**. This value is always equal to half the number of samples specified in `DAQ_Start` or `SCAN_Start` or `Lab_ISCAN_Start` unless the acquisition has not yet begun or the acquisition stopped while in pretrigger mode. In the former case, until NI-DAQ applies an external start trigger, **ptsTfr** is 0. In the latter case (pretrigger mode), the acquisition can stop at any point in the large buffer after acquiring the specified number of samples following a pulse that NI-DAQ applies at STOPTRIG for the MIO-16 or at EXTTRIG* of the AT-MIO-16F-5, AT-MIO-64F-5, and AT-MIO-16X, or at EXTTRIG of Lab and 1200 Series devices. If you are using all E Series devices, see the `Select_Signal` function for information about the external timing signals. Thus, after the acquisition has stopped, the last transfer of data to **strBuffer** contains the number of valid points from the half of the circular buffer where acquisition stopped.

daqStopped is a valid output parameter only if pretrigger mode acquisition is in progress. This parameter indicates the completion of a pretrigger mode acquisition by returning a one (it returns zero otherwise). A one indicates that the acquisition has stopped after taking the specified number of samples following the occurrence of a stop trigger and that NI-DAQ has transferred the last piece of data in the circular buffer to **strBuffer**. The number of data points transferred to **strBuffer**, as always, is equal to **ptsTfr**.

☞ **Note:** ***C Programmers*—ptsTfr *and* daqStopped *are pass-by-reference parameters.***

DAQ_DB_StrTransfer

Continued

Using This Function

Double-buffered data acquisition cyclically fills a buffer with acquired data. The circular buffer is divided into two equal halves so that NI-DAQ can save data from one half while filling the other half. Through the use of a circular buffer, you can collect an unlimited amount of data without requiring an unlimited amount of memory. Double-buffered data acquisition is useful for applications such as writing data to disk and real-time display of data. NI-DAQ can use double-buffered data acquisition for both single-channel and multiple-channel scanned data acquisition. Unless pretrigger mode is in use, you should call `DAQ_Clear` to stop the continuous cyclical double-buffered data acquisition started by `DAQ_Start`, `SCAN_Start`, or `Lab_ISCAN_Start`.

`DAQ_DB_StrTransfer` saves to **strBuffer** one half of the data from the buffer being used for acquisition. An **overWriteError** warning is returned by `DAQ_DB_StrTransfer` if NI-DAQ has overwritten unretrieved data in the circular buffer. Notice that NI-DAQ transfers the data even though it returns the warning. Your application may initiate a double-buffered acquisition and then wait for some condition to be satisfied before beginning retrieval of the data. The first call to the `DB_StrTransfer` function may return the warning because the circular buffer may have *wrapped around* at least once since the acquisition started. If subsequent `DAQ_DB_StrTransfer` calls keep pace with the acquisition, no further **overWriteError** warnings should occur.

`DAQ_DB_StrTransfer` returns an overwrite error if NI-DAQ overwrites data in the half of the circular buffer being copied to the half buffer during the transfer. When this error occurs, the data in the half buffer may be corrupted.

DAQ_DB_Transfer

Format

status = DAQ_DB_Transfer (deviceNumber, halfBuffer, ptsTfr, daqStopped)

Purpose

Transfers half of the data from the buffer being used for double-buffered data acquisition to another buffer, which is passed to the function, and waits until the data to be transferred is available before returning. You can execute `DAQ_DB_Transfer` repeatedly to return sequential half buffers of the data.

Parameters

Input

Name	Type		Description
	Windows	Windows NT	
deviceNumber	I16	I32	assigned by configuration utility

Output

Name	Type		Description
	Windows	Windows NT	
halfBuffer	[I16], HDL	[I16], HDL	integer array to which the data is to be transferred
ptsTfr	U32	U32	number of points transferred
daqStopped	I16	I32	indicates the completion of a pretrigger mode acquisition

DAQ_DB_Transfer

Continued

Parameter Discussion

halfBuffer is an integer array or an `NI_DAQ_Mem` array. The size of the array must be at least half the size of the circular buffer being used for double-buffered data acquisition.

ptsTfr is the number of points transferred to **halfBuffer**. This value is always equal to half the number of samples specified in `DAQ_Start`, `SCAN_Start`, or `Lab_ISCAN_Start` unless the acquisition has not yet begun, or the acquisition stopped while in pretrigger mode. In the former case, until NI-DAQ applies an external start trigger, **ptsTfr** is 0. In the latter case (pretrigger mode), the acquisition can stop at any point in the circular buffer after acquiring the specified number of samples after NI-DAQ applies a pulse at STOPTRIG for the MIO-16 or at EXTTRIG* of the AT-MIO-16F-5, AT-MIO-64F-5, and AT-MIO-16X, or at EXTTRIG of Lab and 1200 Series devices. If you are using all E Series devices, see the `Select_Signal` function for information about the external timing signals. Thus, after the acquisition has stopped, the last transfer of data to **halfBuffer** contains the number of valid points from the half of the circular buffer where acquisition stopped.

daqStopped is a valid output parameter only if pretrigger mode acquisition is in progress. This parameter indicates the completion of a pretrigger mode acquisition by returning a one (it returns zero otherwise). A one indicates that the acquisition has stopped after taking the specified number of samples following the occurrence of a stop trigger, and that NI-DAQ has transferred the last piece of data in the circular buffer to **halfBuffer**. The number of data points transferred to **halfBuffer**, as always, is equal to **ptsTfr**.

Note: ***C Programmers*—ptsTfr *and* daqStopped *must be passed by reference.***

DAQ_Monitor

Format

status = DAQ_Monitor (deviceNumber, channel, sequential, numPts, monitorBuffer, newestPtIndex, daqStopped)

Purpose

Returns data from an asynchronous data acquisition in progress. During a multiple-channel acquisition, you can retrieve data from a single channel or from all channels being scanned. An oldest/newest mode provides for return of sequential (oldest) blocks of data or return of the most recently acquired (newest) blocks of data.

Parameters

Input

Name	Type		Description
	Windows	Windows NT	
deviceNumber	I16	I32	assigned by configuration utility
channel	I16	I32	number of the channel
sequential	I16	I32	enables or disables the return of consecutive or oldest blocks of data
numPts	U32	U32	number of data points you want to retrieve

DAQ_Monitor

Continued

Output

Name	Type		Description
	Windows	Windows NT	
monitorBuffer	[I16], HDL	[I16], HDL	destination buffer for the data
newestPtIndex	U32	U32	offset into the acquisition buffer of the newest point returned
daqStopped	I16	I32	indicates whether the data acquisition has completed

Parameter Discussion

channel is the number of the channel you want to examine. You can choose to set **channel** to a value of -1 to indicate that you want to examine data from all channels being scanned. If **channel** is not equal to -1, **channel** must be equal to the channel selected in `DAQ_Start`, equal to one of the channels selected in `SCAN_Setup`, or equal to one of the channels implied in `Lab_ISCAN_Start`. If you are using an AMUX-64T, **channel** may be equal to any one of the AMUX-64T channels.

Range:	-1 for data from all channels being sampled.
	n where *n* is one of the channels being sampled.

sequential is a flag that enables or disables the return of consecutive or oldest blocks of data from the acquisition buffer. A call to `DAQ_Monitor` with the value of **sequential** equal to one returns a block of data that begins where the last sequential call to `DAQ_Monitor` left off. A call to `DAQ_Monitor` with **sequential** equal to zero returns the most recent block of data available.

0:	Most recent data.
1:	Consecutive data.

numPts is the number of data points you want to retrieve from the buffer being used by the acquisition operation. If the **channel** parameter is equal to -1, **numPts** must be an integer multiple of the number of channels contained in the scan sequence. If you are using one or more AMUX-64T boards, remember that the actual number of channels scanned is equal to the value of the **numChans** parameter you selected in `SCAN_Setup`, multiplied by the number of AMUX-64T boards, multiplied by 4.

DAQ_Monitor

Continued

Range: (if **channel** equals -1) 1 to the value of **count** in the `DAQ_Start`, `SCAN_Start`, or `Lab_ISCAN_Start` call.
(if **channel** is not equal to 1) 1 to the number of points per channel that the acquisition buffer can hold.

monitorBuffer is the destination buffer for the data. It is either an integer array or a `NI_DAQ_Mem` array. **monitorBuffer** must be at least big enough to hold **numPts** worth of data. Upon the return of `DAQ_Monitor`, **monitorBuffer** contains a *snapshot* of a portion of the acquisition buffer.

newestPtIndex is the offset into the acquisition buffer of the newest point returned by `DAQ_Monitor`. When the value of the **sequential** flag is 0, **newestPtIndex** is useful in determining whether you are seeing the same data over and over again. If `DAQ_buffer` is the name of the buffer selected in the `DAQ_Start` call, for example, **monitorBuffer**[**numPts** - 1] = `DAQ_buffer`[**newestPtIndex**], if `DAQ_buffer` is zero based.

daqStopped returns an indication of whether the data acquisition has completed.

0: The DAQ operation is not yet complete.
1: The DAQ operation has completed (or halted due to an error).

Note: ***C Programmers*—newestPtIndex *and* daqStopped *are pass-by-reference parameters.***

Using This Function

`DAQ_Monitor` is intended to return small blocks of data from a background acquisition operation. This is especially useful when you have put the acquisition in a circular mode by enabling either the double-buffered or pretrigger modes. The operation is not disturbed; NI-DAQ merely reads data from the buffer being used by the acquisition. If the amount of data requested is not yet available, `DAQ_Monitor` returns the appropriate error code. Possible uses for `DAQ_Monitor` include deciding when to halt an acquisition based on a level, slope, or peak. `DAQ_Monitor` is also useful for returning data from huge buffers allocated by `NI_DAQ_Mem_Alloc` (because it is not restricted to one-half of the circular buffer size, unlike `DAQ_DB_Transfer`).

If you are using `DAQ_Monitor` to retrieve sequential data (during a circular acquisition) and NI-DAQ overwrites a block of data before it can copy the data, NI-DAQ returns an **overWriteError** warning. `DAQ_Monitor` then restarts sequential retrieval with the most recently acquired block of data.

DAQ_Monitor

Continued

If NI-DAQ overwrites a block of data as it is copied to **monitorBuffer**, NI-DAQ returns the **overWriteError** error. The data in **monitorBuffer** may be corrupted if NI-DAQ returns this error.

DAQ_Op

Format

status = DAQ_Op (deviceNumber, chan, gain, buffer, count, sampleRate)

Purpose

Performs a synchronous, single-channel DAQ operation. `DAQ_Op` does not return until NI-DAQ has acquired all the data or an acquisition error has occurred.

Parameters

Input

Name	Type		Description
	Windows	Windows NT	
deviceNumber	I16	I32	assigned by configuration utility
chan	I16	I32	analog input channel number
gain	I16	I32	gain setting to be used
count	U32	U32	number of samples to be acquired
sampleRate	F64	F64	desired sample rate in units of pts/s

Output

Name	Type		Description
	Windows	Windows NT	
buffer	[I16], HDL	[I16], HDL	contains the acquired data

DAQ_Op

Continued

Parameter Discussion

chan is the analog input channel number. If you are using SCXI, you must use the appropriate analog input channel on the DAQ device that corresponds to the SCXI channel you want. Select the SCXI channel using `SCXI_Single_Chan_Setup` before calling this function. Please refer to the *NI-DAQ User Manual for PC Compatibles* for more information on SCXI channel assignments.

Range: See Table B-1 in Appendix B, *Analog Input Channel and Gain Settings and Voltage Calculation.*

gain is the gain setting to be used for that channel. This gain setting applies only to the DAQ device; if you are using SCXI, you must establish any gain you want at the SCXI module by setting jumpers on the module or by calling `SCXI_Set_Gain`. Refer to Appendix B, *Analog Input Channel and Gain Settings and Voltage Calculation*, for valid gain settings. If you use an invalid gain, NI-DAQ will return an error. NI-DAQ ignores **gain** for 516 and LPM devices and the DAQCard-500/700.

buffer is an integer array or an `NI_DAQ_Mem` array. **buffer** has a length equal to or greater than **count**. When `DAQ_Op` returns with an error number equal to zero, **buffer** contains the acquired data.

count is the number of samples to be acquired (that is, the number of A/D conversions to be performed).

Range: 3 through 2^{32} - 1 (except for the Lab and 1200 Series and E Series devices).
3 through 65,535 (Lab and 1200 Series devices).
2 through 2^{24} (E Series devices).

sampleRate is the sample rate you want in units of pts/s.

Range: Roughly 0.00153 pts/s through 500,000 pts/s. The maximum rate depends on the type of device.

☞ **Note:** ***If you are using an SCXI-1200 with remote SCXI, the maximum rate will depend on the baud rate setting and* count*. Refer to the* SCXI-1200 User Manual *for more details.***

Using This Function

`DAQ_Op` initiates a synchronous process of acquiring A/D conversion samples and storing them in a buffer. `DAQ_Op` does not return control to your application until NI-DAQ acquires all the samples you want (or until an acquisition error occurs). When you are using posttrigger mode (with pretrigger mode disabled), the process stores **count** A/D conversions in the buffer and ignores any subsequent conversions.

Note: ***If you have selected external start triggering of the DAQ operation, a high-to-low edge at the STARTTRIG* I/O connector of the MIO-16, the EXTTRIG* input of the AT-MIO-16F-5, AT-MIO-64F-5 and AT-MIO-16X or a low-to-high edge at the EXTTRIG input of the Lab and 1200 Series devices initiates the DAQ operation. If you are using an E Series device, you need to apply a trigger that you select through the*** `Select_Signal` ***or*** `DAQ_Config` ***functions to initiate data acquisition. Be aware that if you do not apply the start trigger,*** `DAQ_Op` ***does not return control to your application. Otherwise,*** `DAQ_Op` ***issues a software trigger to initiate the DAQ operation.***

If you have enabled pretrigger mode, the sample counter does not begin counting acquisitions until you apply a signal at the stop trigger input. Until you apply this signal, the acquisition remains in a cyclical mode, continually overwriting old data in the buffer with new data. Again, if you do not apply the stop trigger, `DAQ_Op` does not return control to your application.

In any case, you can use `Timeout_Config` to establish a maximum length of time for `DAQ_Op` to execute.

DAQ_Rate

Format

status = DAQ_Rate (rate, units, timebase, sampleInterval)

Purpose

Converts a DAQ rate into the timebase and sample-interval values needed to produce the rate you want.

Parameters

Input

Name	Type		Description
	Windows	Windows NT	
rate	F64	F64	desired DAQ rate
units	I16	I32	pts/s or s/pt (see `CTR_Rate`)

Output

Name	Type		Description
	Windows	Windows NT	
timebase	I16	I32	onboard source signal used
sampleInterval	U16	U32	number of timebase units that elapse between consecutive A/D conversions

Parameter Discussion

rate is the DAQ rate you want. The units in which **rate** is expressed are either points per second (pts/s) or seconds per point (s/pt), depending on the value of the **units** parameter.

Range: Roughly 0.00153 pts/s through 1,000,000 pts/s or 655 s/pt through 0.000001 s/pt.

DAQ_Rate

Continued

units indicates the units used to express **rate**.

0:	pts/s.
1:	s/pt.

timebase is a code representing the resolution of the onboard clock signal that the device uses to produce the acquisition rate you want. Note that this function does not support the 20 MHz clock. You can input the value returned by **timebase** directly to `DAQ_Start`, `SCAN_Start`, `Lab_ISCAN_Start`, or `MDAQ_ScanRate`. **timebase** has the following possible values:

-1:	200 ns (AT-MIO-16F-5, AT-MIO-64F-5, AT-MIO-16X, and EISA-A2000 only).
1:	1 µs.
2:	10 µs.
3:	100 µs.
4:	1 ms.
5:	10 ms.

sampleInterval is the number of timebase units that elapse between consecutive A/D conversions. The combination of the timebase resolution value and the **sampleInterval** produces the DAQ rate you want.

Range: 2 through 65,535.

☞ **Note:** ***C Programmers*—timebase *and* sampleInterval *are pass-by-reference parameters.***

Using This Function

`DAQ_Rate` produces timebase and sample-interval values to closely match the DAQ rate you want. To calculate the actual acquisition rate produced by these values, first determine the clock resolution that
corresponds to the value **timebase** returns. Then use the appropriate formula below, depending on the value specified for units:

units = 0 (pts/s):

actual rate = 1/(clock resolution ∗ **sampleInterval**)

units = 1 (s/pt):

actual rate = clock resolution ∗ **sampleInterval**

DAQ_Start

Format

status = DAQ_Start (deviceNumber, chan, gain, buffer, count, timebase, sampInterval)

Purpose

Initiates an asynchronous, single-channel DAQ operation and stores its input in an array.

Input

Name	Type		Description
	Windows	Windows NT	
deviceNumber	I16	I32	assigned by configuration utility
chan	I16	I32	analog input channel number
gain	I16	I32	gain setting
count	U32	U32	number of samples to be acquired
timebase	I16	I32	timebase value
sampInterval	U16	U32	sample interval

Output

Name	Type		Description
	Windows	Windows NT	
buffer	[I16], HDL	[I16], HDL	used to hold acquired readings

DAQ_Start

Continued

Parameter Discussion

chan is the analog input channel number. If you are using SCXI, you must use the appropriate analog input channel on the DAQ device that corresponds to the SCXI channel you want. Select the SCXI channel using `SCXI_Single_Chan_Setup` before calling this function. Please refer to the *NI-DAQ User Manual for PC Compatibles* for more information on SCXI channel assignments.

Range:	See Table B-1 in Appendix B, *Analog Input Channel and Gain Settings and Voltage Calculation.*

gain is the gain setting to be used for that channel. This gain setting applies only to the DAQ device; if you are using SCXI, you must establish any gain you want at the SCXI module either by setting jumpers on the module or by calling `SCXI_Set_Gain`. Refer to Appendix B, *Analog Input Channel and Gain Settings and Voltage Calculation*, for valid gain settings. If you use invalid gain settings, NI-DAQ returns an error. NI-DAQ ignores **gain** for the 516 and LPM devices and DAQCard-500/700.

buffer is an integer array or an `NI_DAQ_Mem` array. **buffer** must have a length equal to or greater than **count**. The elements of **buffer** are the results of each A/D conversion in the DAQ operation. This buffer is often referred to as the acquisition buffer (or circular buffer when double-buffered mode is enabled) elsewhere in this manual.

count is the number of samples to be acquired (that is, the number of A/D conversions to be performed). For double-buffered acquisitions, **count** must be even.

Range:	3 through 2^{32} - 1 (except a Lab-PC+ that is not enabled for doubled-buffered mode and the E Series devices) 3 through 65,535 (Lab and 1200 Series devices not enabled for double-buffered mode). 2 through 2^{24} (E Series devices).

timebase is the timebase, or resolution, to be used for the sample-interval counter. **timebase** has the following possible values:

-3:	20 MHz clock used as a timebase (50 ns) (E Series only).
-1:	5 MHz clock used as timebase (200 ns resolution) (AT-MIO-16F-5, AT-MIO-64F-5, and AT-MIO-16X only).
0:	External clock used as timebase (connect your own timebase frequency to the internal sample-interval counter via the SOURCE5 input for MIO boards, or, by default, the PFI8 input for E Series devices).
1:	1 MHz clock used as timebase (1 µs resolution).

DAQ_Start

Continued

2:	100 kHz clock used as timebase (10 µs resolution).
3:	10 kHz clock used as timebase (100 µs resolution).
4:	1 kHz clock used as timebase (1 ms resolution).
5:	100 Hz clock used as timebase (10 ms resolution).

On E Series devices, if you use this function with the timebase set at 0, you must call the function `Select_Signal` with signal set to `ND_IN_SCAN_CLOCK_TIMEBASE` (not `ND_IN_CHANNEL_CLOCK_TIMEBASE`), and source set to a value other than `ND_INTERNAL_20_MHZ` and `ND_INTERNAL_100_KHZ` before calling `DAQ_Start` with timebase set to 0; otherwise, DAQ_Start will select low-to-high transitions on the PFI 8 I/O connector pin as your external timebase.

Refer to the *NI-DAQ Function Reference Manual* for further details about the `Select_Signal` function.

If you use external conversion pulses, NI-DAQ ignores the **timebase** parameter and you can set it to any value.

sampInterval indicates the length of the sample interval (that is, the amount of time to elapse between each A/D conversion).
Range: 2 through 65,535.

The sample interval is a function of the timebase resolution. NI-DAQ determines the actual sample interval in seconds using the following formula:

sampInterval ∗ (timebase resolution)

where the timebase resolution for each value of **timebase** is given above. For example, if **sampInterval** = 25 and **timebase** = 2, the sample interval is 25 ∗ 10 µs = 250 µs. If you use external conversion pulses, NI-DAQ ignores the **sampInterval** parameter and you can set it to any value.

☞ **Note:** ***If you are using an SCXI-1200 with remote SCXI, the maximum rate will depend on the baud rate setting and* count*. Refer to the* SCXI-1200 User Manual *for more details.***

Using This Function

`DAQ_Start` configures the analog input multiplexer and gain circuitry as indicated by **chan** and **gain**. If external sample-interval timing has not been indicated by a `DAQ_Config` call, the function sets the sample-interval counter to the specified **sampInterval** and **timebase**. If you have indicated external sample-interval timing, the

DAQ_Start

Continued

DAQ circuitry relies on pulses received on the EXTCONV* input to initiate individual A/D conversions. The sample counter is set up to count the number of samples and to stop the DAQ process when NI-DAQ has acquired **count** samples.

`DAQ_Start` initializes a background process to handle storing of A/D conversion samples into the buffer as NI-DAQ acquires the conversions. When you use posttrigger mode (with pretrigger mode disabled), the process stores up to **count** A/D conversions in the buffer and ignores any subsequent conversions. If a call to `DAQ_Check` returns **status** = 1, the samples are available and NI-DAQ terminates the DAQ process. In addition, a call to `DAQ_Clear` terminates the background DAQ process and enables a subsequent call to `DAQ_Start`. Notice that if `DAQ_Check` returns **daqStopped** = 1 or an error code of -75 or -76, the process is automatically terminated and there is no need to call `DAQ_Clear`.

If you select external start triggering for the DAQ operation, a high-to-low edge at the STARTTRIG* I/O connector input of the MIO-16 and AT-MIO-16D, the EXTTRIG* input of the AT-MIO-16F-5, AT-MIO-64F-5, and AT-MIO-16X, or a low-to-high edge at the EXTTRIG input of Lab and 1200 Series devices initiates the DAQ operation after the `DAQ_Start` call is complete. Otherwise, `DAQ_Start` issues a software trigger to initiate the DAQ operation before returning.

If you have selected external start triggering of the DAQ operation, a high-to-low edge at the STARTTRIG* I/O connector of the MIO-16, the EXTTRIG* input of the AT-MIO-16F-5, a low-to-high edge at the EXTTRIG input of Lab and 1200 Series devices initiates the DAQ operation. If you are using an E Series device, you need to apply a trigger that you select through the `Select_Signal` or `DAQ_Config` functions to initiate data acquisition. Be aware that if you do not apply the start trigger, `DAQ_Op` does not return control to your application. Otherwise, `DAQ_Op` issues a software trigger to initiate the DAQ operation.

If you enable pretrigger mode, the sample counter does not begin counting acquisitions until NI-DAQ applies a signal at the stop trigger input. Until NI-DAQ applies this signal, the acquisition remains in a cyclical mode, continually overwriting old data in the buffer with new data.

☞ **Note:** ***If your application calls*** `DAQ_Start`***,*** `SCAN_Start`***, or*** `Lab_ISCAN_Start`***, always ensure that you call*** `DAQ_Clear` ***before your application terminates and returns control to the operating system. Unpredictable behavior can result unless you make this call (either directly, or indirectly through*** `DAQ_Check` ***or*** `DAQ_DB_Transfer`***).***

DAQ_StopTrigger_Config

Format

status = DAQ_StopTrigger_Config (deviceNumber, stopTrig, ptsAfterStoptrig)

Purpose

Enables the pretrigger mode of data acquisition and indicates the number of data points to acquire after NI-DAQ applies the stop trigger pulse at the STOPTRIG* input of the MIO-16/16D; the EXTTRIG* input of an AT-MIO-16F-5, AT-MIO-64F-5, or AT-MIO-16X or the EXTTRIG input of Lab and 1200 Series devices; or the PFI1 pin of an E Series device. If you are using an E Series device, see the `Select_Signal` description for information about the external timing signals.

Parameters

Input

Name	Type		Description
	Windows	Windows NT	
deviceNumber	I16	I32	assigned by configuration utility
stopTrig	I16	I32	enable or disable the pretriggered mode
ptsAfterStoptrig	U32	U32	number of points to acquire after the trigger

Parameter Discussion

stopTrig indicates whether to enable or disable the pretriggered mode of data acquisition.

0: Disable pretrigger (the default).
1: Enable pretrigger.

ptsAfterStoptrig is the number of data points to acquire after the trigger. This parameter is valid only if **stopTrig** equals 1. For a multiple channel scanned acquisition, **ptsAfterStoptrig** must be an integer multiple of the number of channels scanned.

DAQ_StopTrigger_Config

Continued

Range: 3 through *count*, where *count* is the value of the **count** parameter in the Start call used to start the acquisition. For Lab and 1200 Series devices, the maximum is always 65,535. For an E Series device, the range is 2 through **count**.

Using This Function

Calling `DAQ_StopTrigger_Config` with the **stopTrig** parameter set to 1 causes any subsequent Start call to initiate a cyclical mode data acquisition. In this mode, NI-DAQ writes data continually into your buffer, overwriting data at the beginning of the buffer once NI-DAQ has filled the entire buffer. You can use `DAQ_Check` or `Lab_ISCAN_Check` in this situation to determine where NI-DAQ is currently depositing data in the buffer. Once you apply a pulse at the STOPTRIG* input of the MIO-16/16D or the EXTTRIG* input of the AT-MIO-16F-5, AT-MIO-64F-5, or AT-MIO-16X or the EXTTRIG input of Lab and 1200 Series devices, NI-DAQ acquires an additional number of data points specified by **ptsAfterStoptrig** before the acquisition terminates. `DAQ_Check` or `Lab_ISCAN_Check` will rearrange the data into chronological order (from oldest to newest) and return with the status parameters equal to one when called after termination.

Calling `DAQ_StopTrigger_Config` with **stopTrig** set to 0 returns the acquisition mode to its default, acyclical setting.

You cannot use pretrigger mode in conjunction with the external conversion method on the MIO-16 and AT-MIO-16D.

(E Series devices only) If you use this function with **stopTrig** = 1, the device uses an active high signal from the PFI1 pin as the stop trigger. This selection is consistent with the MIO-16/16D boards. After calling this function, you can use the `Select_Signal` function to take advantage of the DAQ-STC signal routing and polarity selection features.

DAQ_to_Disk

Format

status = DAQ_to_Disk (deviceNumber, chan, gain, filename, count, sampleRate, concat)

Purpose

Performs a synchronous, single-channel DAQ operation and saves the acquired data in a disk file. `DAQ_to_Disk` does not return until NI-DAQ has acquired and saved all the data or an acquisition error has occurred.

Parameters

Input

Name	Type		Description
	Windows	Windows NT	
deviceNumber	I16	I32	assigned by configuration utility
chan	I16	I32	analog input channel number
gain	I16	I32	gain setting
filename	STR	STR	name of data file to be created
count	U32	U32	number of samples to be acquired
sampleRate	F64	F64	rate in units of pts/s
concat	I16	I32	enables concatenation to an existing file

Parameter Discussion

chan is the analog input channel number. If you are using SCXI, you must use the appropriate analog input channel on the DAQ device that corresponds to the SCXI channel you want. Select the SCXI channel using `SCXI_Single_Chan_Setup`

DAQ_to_Disk

Continued

before calling this function. Please refer to the *NI-DAQ User Manual for PC Compatibles* for more information on SCXI channel assignments.

Range:	See Table B-1 in Appendix B, *Analog Input Channel and Gain Settings and Voltage Calculation.*

gain is the gain setting to be used for that channel. This gain setting applies only to the DAQ device; if SCXI is used, you must establish any gain at the SCXI module either by setting jumpers on the module or by calling `SCXI_Set_Gain`. Refer to Appendix B, *Analog Input Channel and Gain Settings and Voltage Calculation*, for valid gain settings. If you use invalid gain settings, NI-DAQ returns an error. NI-DAQ ignores **gain** for 516 and LPM devices and the DAQCard-500/700.

count is the number of samples to be acquired (that is, the number of A/D conversions to be performed). The length of your data file in bytes should be exactly twice the value of **count** upon completion of the acquisition. If you have previously enabled pretrigger mode (by a call to `DAQ_StopTrigger_Config`), NI-DAQ ignores the **count** parameter.

Range:	3 through 2^{32} - 1 (except the E Series devices).
	2 through 2^{24} (E Series devices).

sampleRate is the sample rate you want in units of pts/s.

Range:	Roughly 0.00153 pts/s through 500,000 pts/s. The maximum range varies according to the type of device you have and the speed and degree of fragmentation of your disk storage device.

☞ **Note:** ***If you are using an SCXI-1200 with remote SCXI, the maximum rate will depend on the baud rate setting and* count*. Refer to the* SCXI-1200 User Manual *for more details.***

concat enables concatenation of data to an existing file. Regardless of the value of **concat**, if the file does not exist, it is created.

0:	Overwrite file if it exists.
1:	Concatenate new data to an existing file.

DAQ_to_Disk

Continued

Using This Function

`DAQ_to_Disk` initiates a synchronous process of acquiring A/D conversion samples and storing them in a disk file. `DAQ_to_Disk` does not return control to your application until NI-DAQ acquires and saves all the samples you want (or until an acquisition error occurs).

Note: ***If you select external start triggering for the DAQ operation, a high-to-low edge at the STARTTRIG* I/O connector of the MIO-16 and AT-MIO-16D, the EXTTRIG* input of the AT-MIO-16F-5, AT-MIO-64F-5, and AT-MIO-16X, or a low-to-high edge at the EXTTRIG input of Lab and 1200 Series devices initiates the DAQ operation. If you are using an E Series device, you need to apply a trigger that you select through the*** `Select_Signal` ***or*** `DAQ_Config` ***functions to initiate data acquisition. If you are using all E Series devices, see the*** `Select_Signal` ***function for information about the external timing signals. Be aware that if you do not apply the start trigger,*** `DAQ_to_Disk` ***does not return control to your application. Otherwise,*** `DAQ_to_Disk` ***issues a software trigger to initiate the DAQ operation.***

If you enable pretrigger mode, the sample counter does not begin counting acquisitions until you apply a signal at the stop trigger input. Until you apply this signal, the acquisition continues to write data into the disk file. NI-DAQ ignores the value of the **count** parameter when you enable pretrigger mode. If you do not apply the stop trigger, `DAQ_to_Disk` eventually returns control to your application because, sooner or later, you run out of disk space.

In any case, you can use `Timeout_Config` to establish a maximum length of time for `DAQ_to_Disk` to execute.

DAQ_VScale

Format

status = DAQ_VScale (deviceNumber, chan, gain, gainAdjust, offset, count, binArray, voltArray)

Purpose

Converts the values of an array of acquired binary data and the gain setting for that data to actual input voltages measured.

Parameters

Input

Name	Type		Description
	Windows	Windows NT	
deviceNumber	I16	I32	assigned by configuration utility
chan	I16	I32	channel on which binary reading was taken
gain	I16	I32	gain setting
gainAdjust	F64	F64	multiplying factor to adjust gain
offset	F64	F64	binary offset present in reading
count	U32	U32	length of **binArray** and **voltArray**
binArray	[I16]	[I16]	acquired binary data

DAQ_VScale

Continued

Output

Name	Type		Description
	Windows	Windows NT	
voltArray	[F64]	[F64]	double-precision values returned

Parameter Discussion

chan is the onboard channel or AMUX channel on which the binary data was acquired. For boards other than AT-MIO-16X, AT-MIO-64F-5, and E Series devices, this parameter is ignored because the scaling calculation is the same for all of the channels. However, you are encouraged to pass the correct channel number.

gain is the gain setting at which NI-DAQ acquired the data in **binArray**. If you used SCXI to take the reading, this gain parameter should be the product of the gain on the SCXI module channel and the gain used by the DAQ device.

gainAdjust is the multiplying factor to adjust the gain. Refer to Appendix B, *Analog Input Channel and Gain Settings and Voltage Calculation*, for the procedure for determining **gainAdjust**. If you do not want to do any gain adjustment, (for example, the ideal gain as specified by the parameter **gain**) you must set **gainAdjust** to 1.

offset is the binary offset that needs to be subtracted from **reading**. Refer to Appendix B, *Analog Input Channel and Gain Settings and Voltage Calculation*, for the procedure for determining offset. If you do not want to do any offset compensation, **offset** must be set to zero. The data type is double to allow for offset fractional LSBs. For example, you could use `DAQ_Op` to acquire many samples from a grounded input channel and average them to obtain the offset.

binArray is an array of acquired binary data.

voltArray is an array of double-precision values returned by `DAQ_VScale` and is the voltage representation of **binArray**.

Using This Function

Refer to Appendix B, *Analog Input Channel and Gain Settings and Voltage Calculation*, for the formula used by `DAQ_VScale` to calculate voltage from binary reading.

DIG_Block_Check

Format

status = DIG_Block_Check (deviceNumber, group, remaining)

Purpose

Returns the number of items remaining to be transferred after a `DIG_Block_In` or `DIG_Block_Out` call (DIO-24, AT-MIO-16D, AT-MIO-16DE-10, DIO-32F, PC-DIO-96/PnP, and Lab and 1200 Series devices only).

Parameters

Input

Name	**Type**		**Description**
	Windows	**Windows NT**	
deviceNumber	I16	I32	assigned by configuration utility
group	I16	I32	group

Output

Name	**Type**		**Description**
	Windows	**Windows NT**	
remaining	U32	U32	number of items yet to be transferred

Parameter Discussion

group is the group involved in the asynchronous transfer.

Range:	1 or 2 for most devices.
	1 through 8 for the PC-DIO-96/PnP.

DIG_Block_Check

Continued

remaining is the number of items yet to be transferred. The actual number of bytes remaining to be transferred is equal to **remaining** multiplied by the value of **groupSize** specified in the call to `DIG_Grp_Config` or `DIG_SCAN_Setup`.

☞ **Note:** ***C Programmers:*—remaining *is a pass-by-reference parameter.***

Using This Function

`DIG_Block_Check` monitors an asynchronous transfer of data started via a `DIG_Block_In` or `DIG_Block_Out` call. If NI-DAQ has completed the transfer, `DIG_Block_Check` automatically calls `DIG_Block_Clear`, which permits NI-DAQ to make a new block transfer call immediately.

DIG_Block_Clear

Format

status = DIG_Block_Clear (deviceNumber, group)

Purpose

Halts any ongoing asynchronous transfer, allowing another transfer to be initiated (DIO-24, AT-MIO-16D, AT-MIO-16DE-10, DIO-32F, PC-DIO-96/PnP, and Lab and 1200 series devices only).

Parameters

Input

Name	Type		Description
	Windows	Windows NT	
deviceNumber	I16	I32	assigned by configuration utility
group	I16	I32	group

Parameter Discussion

group is the group involved in the asynchronous transfer.
Range: 1 or 2 for most devices.
1 through 8 for the PC-DIO-96/PnP.

Using This Function

(AT-DIO-32F only) If you aligned the buffer that you used in the previous call to `DIG_Block_Out` or `DIG_Block_In` by a call to `Align_DMA_Buffer`, `DIG_Block_Clear` unaligns that buffer before returning. Unaligning a buffer means that the data is shifted so that the first data point is located at **buffer**[0].

When NI-DAQ has started a block transfer, you must call `DIG_Block_Clear` before NI-DAQ can initiate another block transfer. Notice that `DIG_Block_Check` makes this call for you when it sees that NI-DAQ has completed a transfer. `DIG_Block_Clear` does not change any current group assignments, alter the current handshaking settings, or affect the state of the pattern generation mode.

DIG_Block_In

Format

status = DIG_Block_In (deviceNumber, group, buffer, count)

Purpose

Initiates an asynchronous transfer of data from the specified group to memory (DIO-24, AT-MIO-16D, AT-MIO-16DE-10, DIO-32F, PC-DIO-96/PnP, and Lab and 1200 series devices only).

Parameters

Input

Name	Type		Description
	Windows	Windows NT	
deviceNumber	I16	I32	assigned by configuration utility
group	I16	I32	group
count	U32	U32	number of items to be transferred

Output

Name	Type		Description
	Windows	Windows NT	
buffer	[I16], HDL	[I16], HDL	data obtained by reading the group

Parameter Discussion

group is the group to be read from.

Range:	1 or 2 for most devices.
	1 through 8 for PC-DIO-96/PnP.

DIG_Block_In

Continued

buffer is an integer array or `NI_DAQ_Mem` array that contains the data obtained by reading the group indicated by **group**. For the DIO-32F, NI-DAQ uses all 16 bits in each buffer element. Therefore, the size of the array, in bytes, must be at least **count** multiplied by the size of **group**. For all other devices, only the lower 8 bits of each buffer element are used. Therefore, the size of the array, in bytes, must be at least twice **count** multiplied by the size of **group**.

count is the number of items (for example, 8-bit items for a group of size 1, 16-bit items for a group of size 2, and 32-bit items for a group of size 4) to be transferred to the area of memory specified by **buffer** from the group indicated by **group**.
Range: 2 through 2^{32} - 1.

Using This Function

`DIG_Block_In` initiates an asynchronous transfer of data from a specified group to your buffer. The hardware is responsible for the handshaking details. Call `DIG_Grp_Config` for the DIO-32F or `DIG_SCAN_Setup` for all other devices at least once before calling `DIG_Block_In`. `DIG_Grp_Config` and `DIG_SCAN_Setup` select the group configuration for handshaking.

If you use a DIO-32F, `DIG_Block_In` writes data to all bytes of your buffer regardless of the group size. If the group size is 1, `DIG_Block_In` writes to the lower eight bits of **buffer**[0] on the first read from the group and the upper eight bits of **buffer**[0] on the second read from the group. For example, if the first read returns 0xCD and the second read returns 0xAB, **buffer**[0] is 0xABCD. If group size is 2, `DIG_Block_In` writes data from the lower port (port 0 or port 2) to the lower eight bits of **buffer** [0] and data from the higher port (port 1 or port 3) to the upper eight bits of **buffer** [0]. If group size is 4, `DIG_Block_In` writes the data from ports 0 and 1 to **buffer** [0] and the data from ports 2 and 3 to **buffer** [1].

If you use any device but a DIO-32F, NI-DAQ writes to the lower byte of each buffer element with a value read from the group and sets the upper byte of each buffer element to zero. If the group size is 2, the lower byte of **buffer**[0] receives data from the first port in the group and the lower byte of **buffer**[1] receives data from the second port. NI-DAQ sets the upper bytes of **buffer**[0] and **buffer**[1] to 0.

If you have not configured the specified group as an input group, NI-DAQ does not perform the operation and returns an error. If you have assigned no ports to the specified group, NI-DAQ does not perform the operation and returns an error. You can call `DIG_Block_Check` to monitor the status of a transfer initiated by `DIG_Block_In`.

DIG_Block_In

Continued

For the DIO-32F, pattern generation, if previously enabled, takes effect upon the execution of `DIG_Block_In`. To avoid delays due to DMA reprogramming (DIO-32F boards only), you can use dual DMA (see the `Set_DAQ_Device_Info` function) or you can align your data. If you have aligned your buffer with a call to `Align_DMA_Buffer` and have not called `DIG_Block_Clear` (either directly or indirectly through `DIG_Block_Check`) to unalign the data, you must use the value of

alignIndex returned by `Align_DMA_Buffer` to access your data. In other words, data in an aligned buffer begins at **buffer**[**alignIndex**]. Data in an unaligned buffer begins at **buffer**[0]. See *Pattern Generation I/O with the DIO-32F* in Chapter 3, *Software Overview*, of the *NI-DAQ User Manual for PC Compatibles* for important information about pattern generation.

☞ **Note:** ***Because of a DMA limitation when using the AT-DIO-32F,*** `DIG_Block_In` ***will not work with groups of*** **size** ***= 1.***

☞ **Note:** ***If you are using an SCXI-1200 with Remote SCXI,*** **count** ***is limited by the amount of memory made available on the Remote SCXI unit. For digital buffered input, you are limited to 5,000 bytes of data. The upper bound for*** **count** ***depends on the*** **groupSize** ***set in*** `DIG_SCAN_Setup` ***(for example, if*** **groupSize** ***= 2,*** **count** ***≤ 2,500).***

DIG_Block_Out

Format

status = DIG_Block_Out (deviceNumber, group, buffer, count)

Purpose

Initiates an asynchronous transfer of data from memory to the specified group (DIO-24, AT-MIO-16D, AT-MIO-16DE-10, DIO-32F, PC-DIO-96/PnP, and Lab and 1200 series devices only).

Parameters

Input

Name	Type		Description
	Windows	Windows NT	
deviceNumber	I16	I32	assigned by configuration utility
group	I16	I32	group
buffer	[I16], HDL	[I16], HDL	array containing the user's data
count	U32	U32	number of items to be transferred

Parameter Discussion

group is the group to be written to.
Range: 1 or 2 for most devices.
1 through 8 for PC-DIO-96/PnP.

buffer is an integer array or `NI_DAQ_Mem` array containing your data. NI-DAQ writes the data in this array to the group indicated by **group.** For the DIO-32F, NI-DAQ uses all 16 bits in each buffer element. Therefore, the size of the array, in bytes, must be at least **count** multiplied by the size of **group**. For all other devices, NI-DAQ uses only the lower 8 bits of each buffer element. Therefore, the size of the array, in bytes, must be at least twice **count** multiplied by the size of **group**.

DIG_Block_Out

Continued

count is the number of items (for example, 8-bit items for a group of size 1, 16-bit items for a group of size 2, and 32-bit items for a group of size 4) to be transferred from the area of memory specified by **buffer** to the group indicated by **group**.
Range: 2 through 2^{32} - 1.

Using This Function

`DIG_Block_Out` initiates an asynchronous transfer of data from your buffer to a specified group. The hardware is responsible for the handshaking details. Call `DIG_Grp_Config` for the DIO-32F or `DIG_SCAN_Setup` for the other devices at least once before calling `DIG_Block_Out`. `DIG_Grp_Config` and `DIG_SCAN_Setup` select the group configuration for handshaking.

If you use a DIO-32F, NI-DAQ writes all bytes in your buffer to the group regardless of the group size. If the group size is one, `DIG_Block_Out` writes the lower eight bits of **buffer**[0] to the group on the first write and the upper eight bits of **buffer**[0] to the group on the second write. For example, if **buffer**[0] = 0xABCD, NI-DAQ writes 0xCD to the group on the first write, and writes 0xAB to the group on the second write. If group size is 2, `DIG_Block_Out` writes data from the lower eight bits of **buffer** [0] to the lower port (port 0 or port 2) and data from the upper eight bits of **buffer** [0] to the higher port (port 1 or port 3). If group size is 4, `DIG_Block_Out` writes data from **buffer**[0] to ports 0 and 1 and data from **buffer**[1] to ports 2 and 3.

If you use any device but a DIO-32F, NI-DAQ writes the lower byte of each buffer element to the group in the order indicated in **portList** when you call `DIG_SCAN_Setup`. If the group size is two, on the first write `DIG_Block_Out` writes the lower byte of **buffer**[0] to the first port on **portList** and the lower byte of **buffe**r[1] to the last port on **portList**. For example, if **buffer**[0] = 0xABCD and

buffer[1] is 0x1234, NI-DAQ writes 0xCD to the first port on **portList**, and writes 0x34 to the last port on **portList**.

If you have not configured the specified group as an output group, NI-DAQ does not perform the operation and returns an error. If you have assigned no ports to the specified group, NI-DAQ does not perform the operation and returns an error. You can call `DIG_Block_Check` to monitor the status of a transfer initiated by `DIG_Block_Out`.

If you have previously enabled pattern generation on a DIO-32F, the generation takes effect upon the execution of `DIG_Block_Out`. To avoid delays due to DMA reprogramming, you can use dual DMA (see the `Set_DAQ_Device_Info` function) or you can align your data using the `Align_DMA_Buffer` function. See the *Pattern*

DIG_Block_Out

Continued

Generation I/O with the DIO-32F section in Chapter 3, *Software Overview*, of the *NI-DAQ User Manual for PC Compatibles* for important information about pattern generation.

Note: ***Because of a DMA limitation when using the AT-DIO-32F,*** `DIG_Block_Out` ***will not work with groups of size = 1.***

Note: ***If you are using an SCXI-1200 with Remote SCXI,*** **count** ***is limited by the amount of memory made available on the Remote SCXI unit. For digital buffered output, you are limited to 5,000 bytes of data. The upper bound for*** **count** ***depends on the*** **groupSize** ***set in*** `DIG_SCAN_Setup` ***(for example, if*** **groupSize** ***= 2,*** **count** ***≤ 2,500).***

DIG_Block_PG_Config

Format

status = DIG_Block_PG_Config (deviceNumber, group, config, reqSource, timebase, reqInterval, externalGate)

Purpose

Enables or disables the pattern generation mode of buffered digital I/O (DIO-32F only). When pattern generation is enabled, this function also determines the source of the request signals and, if these are internal, the signal rate and gating mode.

Parameters

Input

Name	Type		Description
	Windows	Windows NT	
deviceNumber	I16	I32	assigned by configuration utility
group	I16	I32	group
config	I16	I32	enables or disables pattern generation
reqSource	I16	I32	source of the request signals
timebase	I16	I32	timebase value
reqInterval	U16	U32	number of **timebase** units between request signals
externalGate	I16	I32	enables or disables external gating

Parameter Discussion

group is the group for which pattern generation is to be enabled or disabled.
Range: 1 or 2.

DIG_Block_PG_Config

Continued

config is a flag that enables or disables pattern generation.

0: Disable pattern generation.

1: Enable pattern generation using double-buffered output (input is always double-buffered).

2: Enable pattern generation using single-buffered output (input is always double-buffered).

reqSource determines the source of the request signals.

0: Internal (from the onboard counters).

1: External (from the signal connected to the REQ pin on the I/O connector).

If request signals are internally generated, nothing must be connected to the REQ pin on the I/O connector. If request signals are externally generated, the signal connected to the REQ pin produces the request pulses (default polarity is active high).

timebase determines the amount of time that elapses during a single **reqInterval**. The following values are possible for **timebase**:

1: 1 µs.

2: 10 µs.

3: 100 µs.

4: 1 ms.

5: 10 ms.

reqInterval is a count of the number of **timebase** units of time that elapses between internally produced request signals.
Range: 2 through 65,535.

externalGate is a flag that enables or disables external gating of an onboard request signal producing counter. If you enable external gating for group 1, the signal connected to IN1 gates the pattern. If you enable external gating for group 2, the signal connected to IN2 gates the pattern. For an AT-DIO-32F, the signal at IN*x* must be high to enable the pattern.

0: Disable external gate.

1: Enable external gate.

Using This Function

`DIG_Block_PG_Config` enables or disables the pattern generation mode of digital I/O. If the **config** parameter equals 1 or 2, any subsequent `DIG_Block_In` or `DIG_Block_Out` call initiates a pattern generation operation. Pattern generation differs from handshaking I/O in that NI-DAQ produces the request signals at regularly

DIG_Block_PG_Config

Continued

clocked intervals. If **reqSource** equals 0, the **timebase** parameter equals 2, and the **reqInterval** parameter equals 10, NI-DAQ reads a new pattern from or writes a pattern to a group every 100 µs.

The advantage of using double-buffered output is that the variability in update intervals is reduced to an absolute minimum, producing the highest quality output at high update rates. The disadvantage is that the first ACK pulse produced by the device is not

preceded by the first pattern. Instead, the second ACK pulse signals the generation of the first pattern. Also, the last pattern generated is not followed by an ACK pulse. The advantage of single-buffered output is the elimination of these ACK pulse irregularities. The first ACK pulse signals generation of the first pattern and the last pattern is followed by a final ACK pulse. The disadvantage of single-buffered output is that at high update rates, variations in DMA bus arbitration times can increase the variability in update intervals, reducing the overall quality of the digital patterns.

DIG_DB_Config

Format

status = DIG_DB_Config (deviceNumber, group, dbMode, oldDataStop, partialTransfer)

Purpose

Enables or disables double-buffered digital transfer operations and sets the double-buffered options (DIO-32F only).

Parameters

Input

Name	Type		Description
	Windows	Windows NT	
deviceNumber	I16	I32	assigned by configuration utility
group	I16	I32	group
dbMode	I16	I32	enable or disable double-buffered mode
oldDataStop	I16	I32	enable or disable regeneration of old data
partialTransfer	I16	I32	enable or disable transfer of final partial half buffer

Parameter Discussion

group is the group to be configured.
Range: 1 or 2.

dbMode indicates whether to enable or disable the double-buffered mode of digital transfer.

0: Disable double buffering (default).
1: Enable double buffering.

DIG_DB_Config

Continued

oldDataStop is a flag whose value enables or disables the mechanism whereby the function stops the digital block output when NI-DAQ is about to output old data a second time. For digital block input, **oldDataStop** enables or disables the mechanism whereby the function stops the input operation before NI-DAQ overwrites unretrieved data.

0:	Allow regeneration of data.
1:	Disallow regeneration of data.

partialTransfer is a flag whose value enables or disables the mechanism whereby NI-DAQ can transfer a final partial half buffer to the digital output block through a `DIG_DB_Transfer` call. The function stops digital block output once NI-DAQ has output the partial half. This field is ignored for input groups.

0:	Disallow partial half buffer transfer.
1:	Allow partial half buffer transfer.

Using This Function

Double-buffered digital block functions cyclically input or output digital data to or from a buffer. The buffer is divided into two equal halves so that NI-DAQ can save or write data from one half while block operations use the other half. For input, this mechanism makes it necessary to alternately save both halves of the buffer so that NI-DAQ does not overwrite data in the buffer before saving the data. For output, the mechanism makes it necessary to alternately write to both halves of the buffer so that NI-DAQ does not output old data. Use `DIG_DB_StrTransfer` or `DIG_DB_Transfer` to save or write the data as NI-DAQ is inputting or outputting the data. You should call `DIG_Clear` to stop the continuous cyclical double-buffered digital operation started by `DIG_Block_Out` or `DIG_Block_In`.

Refer to Chapter 5, *NI-DAQ Double Buffering*, of the *NI-DAQ User Manual for PC Compatibles* for an explanation of double buffering.

Enabling either **oldDataStop** or **partialTransfer** causes an artificial split in the digital block buffer, which requires DMA reprogramming at the end of each half buffer. For a group that is configured for handshaking, this means that a pause in data transfer may occur while NI-DAQ reprograms the DMA. For a group configured for pattern generation, this may cause glitches in the digital input or output pattern (time lapses greater than the programmed period) during DMA reprogramming. Therefore, you should only enable these options if necessary.

DIG_DB_HalfReady

Format

status = DIG_DB_HalfReady (deviceNumber, group, halfReady)

Purpose

Checks whether the next half buffer of data is available during a double-buffered digital block operation. You can use `DIG_DB_HalfReady` to avoid the waiting period that can occur because the double-buffered transfer functions (`DIG_DB_Transfer` and `DIG_DB_StrTransfer`) wait until NI-DAQ can transfer the data before returning.

Parameters

Input

Name	Type		Description
	Windows	Windows NT	
deviceNumber	I16	I32	assigned by configuration utility
group	I16	I32	group

Output

Name	Type		Description
	Windows	Windows NT	
halfReady	I16	I32	whether the next half of data is available

Parameter Discussion

group is the group to be configured.
Range: 1 or 2.

DIG_DB_HalfReady

Continued

halfReady indicates whether the next half buffer of data is available. When **halfReady** equals one, you can use `DIG_DB_Transfer` or `DIG_DB_StrTransfer` to read or write the data immediately. When **halfReady** equals zero, the data is not yet available.

Note: ***C Programmers*—halfReady *is a pass-by-reference parameter.***

Using This Function

Double-buffered digital block functions cyclically input or output digital data to or from a buffer. The buffer is divided into two equal halves so that NI-DAQ can save or write data from one half while block operations use the other half. For input, this mechanism makes it necessary to alternately save both halves of the buffer so that NI-DAQ does not overwrite data in the buffer before saving the data. For output, the mechanism makes it necessary to alternately write to both halves of the buffer so that NI-DAQ does not output old data. Use `DIG_DB_StrTransfer` or `DIG_DB_Transfer` to save or write the data NI-DAQ is inputting or outputting the data. Both of these functions, when called, wait until NI-DAQ can complete the data transfer before returning. During slower paced digital block operations this waiting period can be significant. You can use `DIG_DB_HalfReady` so that the transfer functions are called only when NI-DAQ can make the transfer immediately.

Refer to Chapter 5, *NI-DAQ Double Buffering*, of the *NI-DAQ User Manual for PC Compatibles* for an explanation of double buffering.

DIG_DB_StrTransfer

Format

status = DIG_DB_StrTransfer (deviceNumber, group, strBuffer, ptsTfr)

Purpose

For an input operation, `DIG_DB_StrTransfer` waits until NI-DAQ can transfer half the data from the buffer being used for double-buffered digital block input to a character buffer or a BASIC string, which NI-DAQ passes to the function. For an output operation, `DIG_DB_StrTransfer` waits until NI-DAQ can transfer the data from a character buffer or a BASIC string passed to the function to the buffer being used for double-buffered digital block output. This function is intended for BASIC applications using double-buffered data acquisition where NI-DAQ saves data as it acquires the data. NI-DAQ can then write the string to a disk file using the BASIC `PUT` statement.

Parameters

Input

Name	Type		Description
	Windows	Windows NT	
deviceNumber	I16	I32	assigned by configuration utility
group	I16	I32	group
ptsTfr	U32	U32	points to transfer

Input/Output

Name	Type		Description
	Windows	Windows NT	
strBuffer	[U8]	[U8]	array to which or from which the data is to be transferred

DIG_DB_StrTransfer

Continued

Parameter Discussion

group is the group to be configured.
Range: 1 or 2.

strBuffer is the character array to which or from which NI-DAQ is to transfer the data. The size of the array must be at least half the size of the circular buffer being used for the double-buffered digital block operation.

ptsTfr is only used for output groups with partial transfers enabled. If you have set the partial transfer flag, you can make a transfer to the digital output buffer of less than or equal to half the buffer size, as specified by this field. However, the function will halt the double-buffered digital operation once NI-DAQ makes a transfer of less than half the buffer size. NI-DAQ ignores this field for all other cases (input or output without partial transfers enabled) and the transfer count is equal to half the buffer size.
Range: 0 to half the size of the digital block buffer.

Using This Function

If you have set the partial transfer flag for an output group, the **ptsTfr** field allows NI-DAQ to make transfers of less than half the buffer size to an output buffer. This is useful when NI-DAQ must output a long stream of data but the amount of data is not evenly divisible by half the buffer size. If **ptsTfr** is equal to half the buffer size, the transfer is identical to a transfer without the partial transfer flag set. If **ptsTfr** is less than half the buffer size, however, NI-DAQ makes the transfer to the circular output buffer and alters the DMA reprogramming information so that the digital output operation will halt after NI-DAQ outputs the new data.

Refer to Chapter 5, *NI-DAQ Double Buffering*, of the *NI-DAQ User Manual for PC Compatibles* for an explanation of double buffering and possible error and warning conditions.

DIG_DB_Transfer

Format

status = DIG_DB_Transfer (deviceNumber, group, halfBuffer, ptsTfr)

Purpose

For an input operation, `DIG_DB_Transfer` waits until NI-DAQ can transfer half the data from the buffer being used for double-buffered digital block input to another buffer, which NI-DAQ passes to the function. For an output operation, `DIG_DB_Transfer` waits until NI-DAQ can transfer the data from the buffer passed to the function to the buffer being used for double-buffered digital block output. You can execute `DIG_DB_Transfer` repeatedly to read or write sequential half buffers of data.

Parameters

Input

Name	Type		Description
	Windows	Windows NT	
deviceNumber	I16	I32	assigned by configuration utility
group	I16	I32	group
ptsTfr	U32	U32	points to transfer

Input/Output

Name	Type		Description
	Windows	Windows NT	
halfBuffer	[I16], HDL	[I16], HDL	array to which or from which the data is to be transferred

DIG_DB_Transfer

Continued

Parameter Discussion

group is the group to be configured.
Range: 1 or 2.

halfBuffer is the integer array or `NI_DAQ_Mem` array to which or from which NI-DAQ is to transfer the data. The size of the array must be at least half the size of the circular buffer being used for the double-buffered digital block operation.

ptsTfr is only used for output groups with partial transfers enabled. If you have set the partial transfer flag, NI-DAQ can make a transfer to the digital output buffer of less than or equal to half the buffer size, as specified by this field. However, the function will halt the double-buffered digital operation once NI-DAQ makes a transfer of less than half the buffer size. NI-DAQ ignores this field for all other cases (input or output without partial transfers enabled) and the transfer count is equal to half the buffer size.
Range: 0 to half the size of the digital block buffer.

Using This Function

If you have set the partial transfer flag for an output group, the **ptsTfr** field allows NI-DAQ to make transfers of less than half the buffer size to an output buffer. This is useful when NI-DAQ must output a long stream of data but the amount of data is not evenly divisible by half the buffer size. If **ptsTfr** is equal to half the buffer size, the transfer is identical to a transfer without the partial transfer flag set. If **ptsTfr** is less than half the buffer size, however, NI-DAQ makes the transfer to the circular output buffer and alters the DMA reprogramming information so that the digital output operation will halt after the new data is output.

Refer to Chapter 5, *NI-DAQ Double Buffering*, of the *NI-DAQ User Manual for PC Compatibles* for an explanation of double buffering and possible error and warning conditions.

DIG_Grp_Config

Format

status = DIG_Grp_Config (deviceNumber, group, groupSize, port, dir)

Purpose

Configures the specified group for port assignment, direction (input or output), and size (DIO-32F only).

Parameters

Input

Name	Type		Description
	Windows	Windows NT	
deviceNumber	I16	I32	assigned by configuration utility
group	I16	I32	group
groupSize	I16	I32	size of the group
port	I16	I32	digital I/O port(s) assigned to the group
dir	I16	I32	input or output

Parameter Discussion

group is the group to be configured.
Range: 1 or 2.

groupSize indicates the size of the group. The following values are permitted for **groupSize**:

0:	Unassign any ports previously assigned to **group**.
1:	One port assigned (8-bit group) to **group**.
2:	Two ports assigned (16-bit group) to **group**.
4:	Four ports assigned (32-bit group) to **group**.

DIG_Grp_Config

Continued

port indicates the digital I/O port or ports assigned to the group. The assignments made depend on the values of **port** and of **groupSize**:

groupSize = 1	**port** = 0 assigns port 0 to group 1.
	port = 1 assigns port 1 to group 1.
	port = 2 assigns port 2 to group 2.
	port = 3 assigns port 3 to group 2.
groupSize = 2	**port** = 0 assigns ports 0 and 1 to group 1.
	port = 2 assigns ports 2 and 3 to group 2.
groupSize = 4	**port** = 0 assigns ports 0, 1, 2, and 3 to group 1.

dir indicates the direction, input, or output for which the group is to be configured.

0:	Port is configured as an input port (default).
1:	Port is configured as an output port.
3:	Port is configured as an input port with the double-buffering hardware latch disabled.
4:	Port is configured as an output port with the double-buffering hardware on.

Using This Function

`DIG_Grp_Config` configures the specified group according to the port assignment and direction. If **groupSize** = 0, NI-DAQ releases any ports assigned to the group specified by **group** and clears the group handshake circuitry. If **groupSize** = 1, 2, or 4, NI-DAQ assigns the specified ports to the group and configures the ports for the specified direction. NI-DAQ subsequently writes to or reads from ports assigned to a group using the `DIG_In_Grp` and `DIG_Out_Grp` or the `DIG_Block_In` and `DIG_Block_Out` functions. NI-DAQ can no longer access any ports assigned to a group through any of the nongroup calls listed previously. Only the `DIG_Block` calls can use a group of size 4.

If you are using an AT-DIO-32F and intend to perform block I/O, you are limited to group sizes of 2 and 4. After system startup, no ports are assigned to groups.

See the *AT-DIO-32F User Manual* for group handshake timing.

DIG_Grp_Mode

Format

status = DIG_Grp_Mode (deviceNumber, group, signal, edge, reqPol, ackPol, ackDelayTime)

Purpose

Configures the specified group for handshake signal modes (DIO-32F only).

Parameters

Input

Name	Type		Description
	Windows	Windows NT	
deviceNumber	I16	I32	assigned by configuration utility
group	I16	I32	group
signal	I16	I32	level or pulsed (edge-triggered) handshake signals
edge	I16	I32	rising-edge or falling-edge pulsed signals
reqPol	I16	I32	request signal is to be active high or active low
ackPol	I16	I32	acknowledge handshake signal is to be active high or active low
ackDelayTime	I16	I32	data settling time allowed

DIG_Grp_Mode

Continued

Parameter Discussion

group is the group to be configured.
Range: 1 or 2.

signal indicates whether the group is to be configured for level or pulsed (edge-triggered) handshake signals.

0:	Group is configured for level handshake signals.
1:	Group is configured for pulsed handshake signals.
2:	Group is configured for pulsed handshake signals with variable ACK pulse width.

☞ **Note:** ***This function does not support variable-length ACK pulse width (signal = 2) on AT-DIO-32F Revision B (and earlier).***

edge indicates whether the group is to be configured for leading-edge or trailing-edge pulsed signals. **edge** is valid only if **signal** = 1 or 2.

0:	Group is configured for leading-edge pulsed handshake signals.
1:	Group is configured for trailing-edge pulsed handshake signals. This setting does not support variable ACK pulse width (**signal** = 2).

reqPol indicates whether the group request signal is to be active high or active low.

0:	Group is configured for active high (non-inverted) request handshake signal polarity.
1:	Group is configured for active low (inverted) request handshake signal polarity.

ackPol indicates whether the group acknowledge handshake signal is to be active high or active low.

0:	Group is configured for active high (non-inverted) acknowledge handshake signal polarity.
1:	Group is configured for active low (inverted) acknowledge handshake signal polarity.

ackDelayTime indicates the data settling time allowed for the group. If **signal** equals 0 or 1, the value of **ackDelayTime** specifies the number of 100 ns intervals that elapse until NI-DAQ generates the ACK signal. If **signal** is 2, the value of **ackDelayTime** specifies the duration of the ACK pulse.
Range: 0 through 7.

0:	No settling time.
7:	700 ns settling time.

DIG_Grp_Mode

Continued

Using This Function

`DIG_Grp_Mode` configures the group handshake signals according to the specified parameters with respect to the specified port assignment and direction. After initialization, the default handshake mode for each group is as follows:
signal = 0: Level handshake signals.
edge = 0: **edge** parameter not valid because **signal** = 0.
reqPol = 0: Request handshake signal is not inverted (active high).
ackPol = 0: Acknowledge handshake signal is not inverted (active high).
ackDelayTime = 0: Settling time is 0 ns.

You need to call `DIG_Grp_Mode` only if you need different handshake modes. Refer to the *AT-DIO-32F User Manual* for handshake timing and mode information.

☞ **Note:** ***(AT-DIO-32F Revision B boards only) Do not use a leading-edge, pulsed handshaking signal for an input group. NI-DAQ cannot latch the data into the port in this mode, and, if new data is presented to the port before NI-DAQ reads and saves the old data, the old data is lost.***

DIG_Grp_Status

Format

status = DIG_Grp_Status (deviceNumber, group, latched)

Purpose

Returns a status word indicating the handshake status of the specified group (DIO-32F only).

Parameters

Input

Name	Type		Description
	Windows	Windows NT	
deviceNumber	I16	I32	assigned by configuration utility
group	I16	I32	group

Output

Name	Type		Description
	Windows	Windows NT	
latched	I16	I32	handshake status

Parameter Discussion

group is the group whose handshake status is to be obtained.
Range: 1 or 2.

latched returns the handshake status of the group. **latched** can be either 0 or 1. The significance of **latched** depends on the configuration of the group. If the group is configured as an input group, **latched** = 1 indicates that NI-DAQ has latched data into the ports making up that group. If the group is configured as an output group,

DIG_Grp_Status

Continued

latched = 1 indicates that an external device has latched the output of the ports making up the group and that NI-DAQ can write new data to the group.

Note: ***C Programmers*—latched *is a pass-by-reference parameter.***

Using This Function

`DIG_Grp_Status` reads the handshake status of the specified group and returns an indication of the group status in **latched**. `DIG_Grp_Status`, along with `DIG_Out_Grp` and `DIG_In_Grp`, facilitates handshaking of digital data between systems. If the specified group is configured as an input group and `DIG_Grp_Status` returns **latched** = 1, `DIG_In_Grp` can fetch the data an external device has latched in. If the specified group is configured as an output group and `DIG_Prt_Status` returns **latched** = 1, `DIG_Out_Grp` can write the next piece of data to the external device. If the specified group is not assigned any ports, NI-DAQ returns an error code and **latched** = 0.

You must call `DIG_Grp_Config` to assign ports to a group and to configure a group for data direction. Group configuration is discussed under the `DIG_Grp_Config` description.

The state of **latched** corresponds to the state of the DIO-32F DRDY bit. Refer to the *AT-DIO-32F User Manual* for handshake timing details.

DIG_In_Grp

Format

status = DIG_In_Grp (deviceNumber, group, groupPattern)

Purpose

Reads digital input data from the specified digital group (DIO-32F only).

Parameters

Input

Name	Type		Description
	Windows	Windows NT	
deviceNumber	I16	I32	assigned by configuration utility
group	I16	I32	group

Output

Name	Type		Description
	Windows	Windows NT	
groupPattern	I16	I16	digital data read from the ports

Parameter Discussion

group is the group to be read from.
Range: 1 or 2.

DIG_In_Grp

Continued

groupPattern returns the digital data read from the ports in the specified group. **groupPattern** is mapped to the digital input ports making up the group in the following way:

- If the group contains one port, NI-DAQ returns the eight bits read from that port in the low-order eight bits of **groupPattern**.
- If the group contains two ports, NI-DAQ returns the 16 bits read from those ports in the following way: if the group contains ports 0 and 1, NI-DAQ returns the value read from port 0 in the low-order eight bits, and NI-DAQ returns the value read from port 1 in the high-order eight bits. If the group contains ports 2 and 3, NI-DAQ returns the value read from port 2 in the low-order eight bits, and NI-DAQ returns the value read from port 3 in the high-order eight bits. NI-DAQ reads from the two ports simultaneously.
- If the group contains four ports, NI-DAQ returns a **deviceSupportError**.

Note: ***C Programmers*—groupPattern *is a pass-by-reference parameter.***

Using This Function

`DIG_In_Grp` returns digital data from the group on the specified device. If the group is configured as an input group, reading that group returns the digital logic state of the lines of the ports in the group as some external device is driving them. If the group is configured as an output group and has read-back capability, reading the group returns the output state of that group. If no ports have been assigned to the group, NI-DAQ does not perform the operation and returns an error code. You must call `DIG_Grp_Config` to assign ports to a group and to configure the group as an input or output group.

DIG_In_Line

Format

status = DIG_In_Line (deviceNumber, port, line, state)

Purpose

Returns the digital logic state of the specified digital line in the specified port.

Parameters

Input

Name	Type		Description
	Windows	Windows NT	
deviceNumber	I16	I32	assigned by configuration utility
port	I16	I32	digital I/O port number
line	I16	I32	digital line to be read

Output

Name	Type		Description
	Windows	Windows NT	
state	I16	I32	returns the digital logic state

DIG_In_Line

Continued

Parameter Discussion

port is the digital I/O port number.

Range:	0 through 1 for the AT-AO-6/10, DAQCard-500/700, PC-TIO-10, PC-OPDIO-16, and 516, AO-2DC, Am9513-based, and LPM devices.
	0 for the E Series devices, except the AT-MIO-16DE-10.
	0 through 2 for the DIO-24 and Lab and 1200 series devices.
	0 and 2 through 4 for the AT-MIO-16DE-10.
	0 through 4 for the DIO-32F and AT-MIO-16D.
	0 through 11 for the PC-DIO-96/PnP.

line is the digital line to be read.

Range: 0 through *k*-1, where *k* is the number of digital I/O lines making up the port.

state returns the digital logic state of the specified line.

0:	The specified digital line is at a digital logic low.
1:	The specified digital line is at a digital logic high.

☞ **Note:** ***C Programmers*—state *is a pass-by-reference parameter.***

Using This Function

`DIG_In_Line` returns the digital logic state of the specified digital line in the specified port. If the specified port is configured as an input port, NI-DAQ determines the state of the specified line by the way in which some external device is driving it. If the port is configured as an output port and the port has read-back capability, NI-DAQ determines the state of the line by the way in which that port itself is driving it. Reading a line configured for output on the PC-TIO-10 or an E Series device returns a warning stating that NI-DAQ has read an output line.

DIG_In_Port

Format

status = DIG_In_Port (deviceNumber, port, pattern)

Purpose

Returns digital input data from the specified digital I/O port.

Parameters

Input

Name	Type		Description
	Windows	Windows NT	
deviceNumber	I16	I32	assigned by configuration utility
port	I16	I32	digital I/O port number

Output

Name	Type		Description
	Windows	Windows NT	
pattern	I16	I16	8-bit digital data read from the specified port

Parameter Discussion

port is the digital I/O port number.

Range: 0 through 1 for the AT-AO-6/10, DAQCard-500/700, PC-TIO-10, PC-OPDIO-16, and 516, AO-2DC, Am9513-based, and LPM devices.
0 for the E Series devices, except the AT-MIO-16DE-10.
0 through 2 for the DIO-24 and Lab and 1200 series devices.
0 and 2 through 4 for the AT-MIO-16DE-10.
0 through 4 for the DIO-32F and AT-MIO-16D.
0 through 11 for the PC-DIO-96/PnP.

DIG_In_Port

Continued

pattern returns the 8-bit digital data read from the specified port. NI-DAQ maps **pattern** to the digital input lines making up the port such that bit 0, the least significant bit, corresponds to digital input line 0. The high eight bits of **pattern** are always 0. If the port is less than eight bits wide, NI-DAQ also sets the bits in the low-order byte of **pattern** that do not correspond to lines in the port to 0. For example, because ports 0 and 1 on the Am9513-based boards are four bits wide, only bits 0 through 3 of **pattern** reflect the digital state of these ports, while NI-DAQ sets all other bits of **pattern** to 0.

☞ **Note:** ***C Programmers*—pattern *is a pass-by-reference parameter.***

Using This Function

`DIG_In_Port` reads digital data from the port on the specified device. If the port is configured as an input port, reading that port returns the digital logic state of the lines as some external device is driving them. If the port is configured as an output port and has read-back capability, reading the port returns the output state of that port, along with a warning that NI-DAQ has read an output port.

DIG_Line_Config

Format

status = DIG_Line_Config (deviceNumber, port, line, dir)

Purpose

Configures a specific line on a port for direction (input or output).

Parameters

Input

Name	Type		Description
	Windows	Windows NT	
deviceNumber	I16	I32	assigned by configuration utility
port	I16	I32	digital I/O port number
line	I16	I32	digital line
dir	I16	I32	direction, input, or output

Parameter Discussion

port is the digital I/O port number.

Range:	0 for the E Series devices.
	0 through 1 for the PC-TIO-10.

line is the digital line for which to configure.

Range: 0 through 7.

dir indicates the direction, input or output, to which the line is to be configured.

0:	Line is configured as an input line (default).
1:	Line is configured as an output line.

DIG_Line_Config

Continued

Using This Function

With this function, a PC-TIO-10 or E Series port can have any combination of input and output lines. Use `DIG_Prt_Config` to set all lines on the port to be either all input or all output lines.

DIG_Out_Grp

Format

status = DIG_Out_Grp (deviceNumber, group, groupPattern)

Purpose

Writes digital output data to the specified digital group (DIO-32F only).

Parameters

Input

Name	Type		Description
	Windows	Windows NT	
deviceNumber	I16	I32	assigned by configuration utility
group	I16	I32	group
groupPattern	I16	I16	digital data to be written

Parameter Discussion

group is the group to be written to.
Range: 1 or 2.

groupPattern is the digital data to be written to the specified port. NI-DAQ maps **groupPattern** to the digital output ports making up the group in the following way:

- If the group contains one port, NI-DAQ writes the low-order eight bits of **groupPattern** to that port.
- If the group contains two ports, NI-DAQ writes all 16 bits of **groupPattern** to those ports. If the group contains ports 0 and 1, NI-DAQ writes the low-order eight bits to port 0 and the high-order eight bits to port 1. If the group contains ports 2 and 3, NI-DAQ writes the low-order eight bits to port 2 and the high-order eight bits to port 3. NI-DAQ writes to the two ports simultaneously.
- If the group contains four ports, NI-DAQ returns a **deviceSupportError**.

DIG_Out_Grp

Continued

Using This Function

`DIG_Out_Grp` writes the specified digital data to the group on the specified device. If you have not configured the specified group as an output group, NI-DAQ does not perform the operation and returns an error. If you have assigned no ports to the specified group, NI-DAQ does not perform the operation and returns an error. You must call `DIG_Grp_Config` to configure a group.

DIG_Out_Line

Format

status = DIG_Out_Line (deviceNumber, port, line, state)

Purpose

Sets or clears the specified digital output line in the specified digital port.

Parameters

Input

Name	Type		Description
	Windows	Windows NT	
deviceNumber	I16	I32	assigned by configuration utility
port	I16	I32	digital I/O port number
line	I16	I32	digital output line
state	I16	I32	new digital logic state

Parameter Discussion

port is the digital I/O port number.

Range:	0 through 1 for the AT-AO-6/10, DAQCard-500/700, PC-TIO-10, PC-OPDIO-16, and 516, AO-2DC, Am9513-based, and LPM devices.
	0 for the E Series devices, except the AT-MIO-16DE-10.
	0 through 2 for the DIO-24 and Lab and 1200 series devices.
	0 and 2 through 4 for the AT-MIO-16DE-10.
	0 through 4 for the DIO-32F and AT-MIO-16D.
	0 through 11 for the PC-DIO-96/PnP.

line is the digital output line to be written to.

Range:	0 through *k*-1, where *k* is the number of digital I/O lines making up the port.

DIG_Out_Line

Continued

state contains the new digital logic state of the specified line.

0:	The specified digital line is set to digital logic low.
1:	The specified digital line is set to digital logic high.

Using This Function

`DIG_Out_Line` sets the digital line in the specified port to the specified state. The remaining digital output lines making up the port are not affected by this call. If you have not configured the port as an output port, NI-DAQ does not perform the operation and returns an error. Except for the PC-TIO-10 or an E Series device, you must call `DIG_Prt_Config` to configure a digital I/O port as an output port. On the PC-TIO-10 or an E Series device, you need only configure the specified line for output using `DIG_Prt_Config` or `DIG_Line_Config`.

☞ **Note:** ***Connecting one or more AMUX-64T boards or an SCXI chassis to an MIO or AI device causes*** `DIG_Out_Line` ***to return a*** **badInputValErr** ***when called with port equal to 0 and line equal to one of the following values:***

One AMUX-64T device—line equal to 0 or 1.
Two AMUX-64T boards—line equal to 0, 1, or 2.
Four AMUX-64T boards—line equal to 0, 1, 2, or 3.
An SCXI chassis—line equal to 0, 1, or 2 (and 4 for the E Series devices only).

DIG_Out_Port

Format

status = DIG_Out_Port (deviceNumber, port, pattern)

Purpose

Writes digital output data to the specified digital port.

Parameters

Input

Name	Type		Description
	Windows	Windows NT	
deviceNumber	I16	I32	assigned by configuration utility
port	I16	I32	digital I/O port number
pattern	I16	I16	8-bit digital pattern for the data written

Parameter Discussion

port is the digital I/O port number.

Range: 0 through 1 for the AT-AO-6/10, DAQCard-500/700, PC-TIO-10, PC-OPDIO-16, and 516, AO-2DC, Am9513-based, and LPM devices.
0 for the E Series devices, except the AT-MIO-16DE-10.
0 through 2 for the DIO-24 and Lab and 1200 series devices.
0 and 2 through 4 for the AT-MIO-16DE-10.
0 through 4 for the DIO-32F and AT-MIO-16D.
0 through 11 for the PC-DIO-96/PnP.

pattern indicates the 8-bit digital pattern for the data written to the specified port. NI-DAQ ignores the high eight bits of **pattern**. NI-DAQ maps the low eight bits of **pattern** to the digital output lines making up the port so that bit 0, the least significant bit, corresponds to digital output line 0. If the port is less than eight bits wide, only the low-order bits in **pattern** affect the port. For example, because ports 0 and 1 on the

DIG_Out_Port

Continued

Am9513-based boards are four bits wide, only bits 0 through 3 of **pattern** affect the digital output state of these ports.

Using This Function

`DIG_Out_Port` writes the specified digital data to the port on the specified device. If you have not configured the specified port as an output port, NI-DAQ does not perform the operation and returns an error. You must call `DIG_Prt_Config` to configure a digital I/O port as an output port. Using `DIG_Out_Port` on a port with a combination of input and output lines returns a warning that some lines are configured for input.

☞ **Note:** ***If you have connected one or more AMUX-64T boards or an SCXI chassis to your Am9513-based boards,*** `DIG_Out_Port` ***returns a*** **badPortErr** ***if called with port equal to 0.***

DIG_Prt_Config

Format

status = DIG_Prt_Config (deviceNumber, port, mode, dir)

Purpose

Configures the specified port for direction (input or output). `DIG_Prt_Config` also sets the handshake mode for the DIO-24, AT-MIO-16D, AT-MIO-16DE-10, PC-DIO-96/PnP, and Lab and 1200 series devices.

Parameters

Input

Name	Type		Description
	Windows	Windows NT	
deviceNumber	I16	I32	assigned by configuration utility
port	I16	I32	digital I/O port number
mode	I16	I32	handshake mode
dir	I16	I32	direction, input, or output

Parameter Discussion

port is the digital I/O port number.

Range:	0 through 1 for the AT-AO-6/10, DAQCard-500/700, PC-TIO-10, PC-OPDIO-16, and 516, AO-2DC, Am9513-based, and LPM devices.
	0 for the E Series devices, except the AT-MIO-16DE-10.
	0 through 2 for the DIO-24 and Lab and 1200 series devices.
	0 through 3 for the DIO-32F.
	0 and 2 through 4 for the AT-MIO-16DE-10.
	0 through 4 for the AT-MIO-16D.
	0 through 11 for the PC-DIO-96/PnP.

mode indicates the handshake mode that the port uses.

DIG_Prt_Config

Continued

0: Port is configured for no-handshaking (nonlatched) mode. You must use **mode** = 0 for all other ports and boards. You can use the DIO-32F for handshaking, but only through the group calls (see `DIG_Grp_Config`).

1: Port is configured for handshaking (latched) mode. **mode** = 1 is valid only for ports 0 and 1 of the DIO-24 and Lab and 1200 series devices; for ports 2 and 3 of the AT-MIO-16D and AT-MIO-16DE-10; and for ports 0, 1, 3, 4, 6, 7, 9, and 10 of the PC-DIO-96/PnP.

dir indicates the direction, input or output, to which the port is to be configured.

0: Port is configured as an input port (default).

1: Port is configured as an output port.

2: Port is configured as a bidirectional port.

The following ports can be configured as bidirectional:

Device	**Ports**
AT-MIO-16D	2
AT-MIO-16DE-10	2
Lab and 1200 series devices	0
DIO-24	0
PC-DIO-96/PnP	0, 3, 6, and 9

Using This Function

`DIG_Prt_Config` configures the specified port according to the specified direction and handshake mode. Any configurations not supported by or invalid for the specified port return an error, and NI-DAQ does not change the port configuration. Information about the valid configuration of any digital I/O port is in Chapter 2, *Hardware Overview*, and Chapter 3, *Software Overview*, of the *NI-DAQ User Manual for PC Compatibles*.

For the DIO-24, AT-MIO-16D, DIO-32F, PC-DIO-96/PnP, and Lab and 1200 series devices, `DIG_Prt_Config` returns an error if the specified port has been assigned to a group by a previous call to `DIG_Grp_Config` or `DIG_SCAN_Setup`. `DIG_Prt_Config` also returns an error for the DIO-32F if the specified port is port 4.

DIG_Prt_Config

Continued

After system startup, the digital I/O ports on all the boards supported by this function are configured as follows:
dir = 0: Input port.
mode = 0: No-handshaking mode.

Also, ports on the DIO-24, AT-MIO-16D, DIO-32F, PC-DIO-96/PnP, and Lab and 1200 series devices are not assigned to any group. If this is not the digital I/O configuration you want, you must call `DIG_Prt_Config` to change the port configuration. You must

call `DIG_Grp_Config` to use handshaking modes on the DIO-32F. Configuring any port on boards that use the 8255 chip will cause the output state of the other digital ports to change. Therefore, you should configure all ports before outputting data.

☞ **Note:** ***AT-MIO-16D, AT-MIO-16DE-10, Lab and 1200 series, PC-AO-2DC, PC-DIO-24, and PC-DIO-96/PnP users—Because of the design of the Intel 8255 chip, calling this function on one port will reset the output states of lines on other ports on the same 8255 chip. The other ports will remain in the same configuration; input ports are not affected.***

☞ **Note:** ***If you have connected one or more AMUX-64T boards or an SCXI chassis to your MIO or AI device,*** `DIG_Prt_Config` ***returns a*** **badPortErr** ***if called with port equal to 0.***

DIG_Prt_Status

Format

status = DIG_Prt_Status (deviceNumber, port, latched)

Purpose

Returns a status word indicating the handshake status of the specified port (DIO-24, AT-MIO-16D, AT-MIO-16DE-10, PC-DIO-96/PnP, and Lab and 1200 series devices only).

Parameters

Input

Name	Type		Description
	Windows	Windows NT	
deviceNumber	I16	I32	assigned by configuration utility
port	I16	I32	digital I/O port number

Output

Name	Type		Description
	Windows	Windows NT	
latched	I16	I32	handshake status

Parameter Discussion

port is the digital I/O port number.

Range:	0 or 1 for the DIO-24 and Lab and 1200 series devices.
	2 or 3 for the AT-MIO-16D and AT-MIO-16DE-10.
	0, 1, 3, 4, 6, 7, 9, and 10 for the PC-DIO-96/PnP.

DIG_Prt_Status

Continued

latched returns the handshake status of the port.

0:	A port is not available for reading from an input port or writing to an output port.
1:	A unidirectional port is available for reading from an input port or writing to an output port.
2:	A bidirectional port is ready for reading.
3:	A bidirectional port is ready for writing.
4:	A bidirectional port is ready for reading and writing.

☞ **Note:** ***C Programmers*—latched *is a pass-by-reference parameter.***

Using This Function

`DIG_Prt_Status` reads the handshake status of the specified port and returns the port status in **latched**. `DIG_Prt_Status`, along with `DIG_Out_Port` and `DIG_In_Port`, facilitates handshaking of digital data between systems. If the specified port is configured as an input port, `DIG_Prt_Status` indicates when to call `DIG_In_Port` to fetch the data that an external device has latched in. If the specified port is configured as an output port, `DIG_Prt_Status` indicates when to call `DIG_Out_Port` to write the next piece of data to the external device. If the specified port is not configured for handshaking, NI-DAQ returns an error code and **latched** = 0.

Refer to your device user manual for handshake timing information. If the port is configured for input handshaking, **latched** corresponds to the state of the IBF bit. If the port is configured for output handshaking, **latched** corresponds to the state of the OBF* bit.

☞ **Note:** ***You must call*** `DIG_Prt_Config` ***to configure a port for data direction and handshaking operation.***

DIG_SCAN_Setup

Format

status = DIG_SCAN_Setup (deviceNumber, group, groupSize, portList, dir)

Purpose

Configures the specified group for port assignment, direction (input or output), and size (DIO-24, AT-MIO-16D, AT-MIO-16DE-10, PC-DIO-96/PnP, and Lab and 1200 series devices only).

Parameters

Input

Name	Type		Description
	Windows	Windows NT	
deviceNumber	I16	I32	assigned by configuration utility
group	I16	I32	group to be configured
groupSize	I16	I32	number of 8-bit ports
portList	[I16]	[I32]	list of ports
dir	I16	I32	direction, input, or output

DIG_SCAN_Setup

Continued

Parameter Discussion

group is the group to be configured.

Range:	1 through 2 for most devices.
	1 through 8 for the PC-DIO-96/PnP.

groupSize selects the number of 8-bit ports in the group.

Range:	0 through 2 for most devices.
	0 through 8 for the PC-DIO-96/PnP.

☞ **Note:** ***Zero is to unassign any ports previously assigned to group.***

portList is the list of ports in **group**. The order of the ports in the list determines how NI-DAQ interleaves data in your buffer when you call `DIG_Block_In` or `DIG_Block_Out`. The last port in the list determines the port whose handshaking signal lines NI-DAQ uses to communicate with the external device and to generate hardware interrupt.

Range:	0 or 1 for most devices.
	2 or 3 for the AT-MIO-16D and AT-MIO-16DE-10.
	0, 1, 3, 4, 6, 7, 9, or 10 for the PC-DIO-96/PnP.

dir selects the direction, input or output, to which the **group** is to be configured.

0:	Port is configured as an input port (default).
1:	Port is configured as an output port.
2:	Port is configured as a bidirectional port.

The following ports can be configured as bidirectional:

Device	**Ports**
AT-MIO-16D	2
AT-MIO-16DE-10	2
Lab and 1200 series devices	0
DIO-24	0
PC-DIO-96/PnP	0, 3, 6, and 9

DIG_SCAN_Setup

Continued

Using This Function

`DIG_SCAN_Setup` configures the specified group according to the specified port assignment and direction. If **groupSize** is 0, NI-DAQ releases any ports previously assigned to **group**. Any configurations not supported by or invalid for the specified group return an error, and NI-DAQ does not change the group configuration. NI-DAQ subsequently writes to or reads from ports assigned to a group as a group using `DIG_Block_In` and `DIG_Block_Out`. NI-DAQ can no longer access any ports assigned to a group through any of the non-group calls listed previously.

Because each port on the DIO-24, AT-MIO-16D, AT-MIO-16DE-10, and Lab and 1200 series devices has its own handshaking circuitry, extra wiring may be necessary to make data transfer of a group with more than one port reliable. If the group has only one port, no extra wiring is needed.

Each input port has a different Strobe Input (STB*) control signal.

- PC4 on the I/O connector is for port 0.
- PC2 on the I/O connector is for port 1.

Each input port also has a different Input Buffer Full (IBF) control signal.

- PC5 on the I/O connector is for port 0.
- PC1 on the I/O connector is for port 1.

Each output port has a different Output Buffer Full (OBF*) control signal.

- PC7 on the I/O connector is for port 0.
- PC1 on the I/O connector is for port 1.

Each output port also has a different Acknowledge Input (ACK*) control signal.

- PC6 on the I/O connector is for port 0.
- PC2 on the I/O connector is for port 1.

On the PC-DIO-96/PnP I/O connector, you can find four different sets of PC pins. They are APC, BPC, CPC, and DPC. APC pins correspond to port 0 and port 1, BPC pins correspond to port 3 and port 4, CPC pins correspond to port 6 and port 7, and DPC pins correspond to port 9 and port 10. For example, CPC7 is the Output Buffer Full (OBF) control signal for port 6 and CPC1 is the Output Buffer Full (OBF) for port 7 if both ports are configured as handshaking output ports.

DIG_SCAN_Setup

Continued

If a group of ports is configured as input, you need to tie all the corresponding Strobe Input (STB*) together and connect them to the appropriate handshaking signal of the external device. You should connect only the Input Buffer Full (IBF) of the last port on **portList** to the external device. No connection is needed for the IBF of the other port on **portList**. See Figure 2-12.

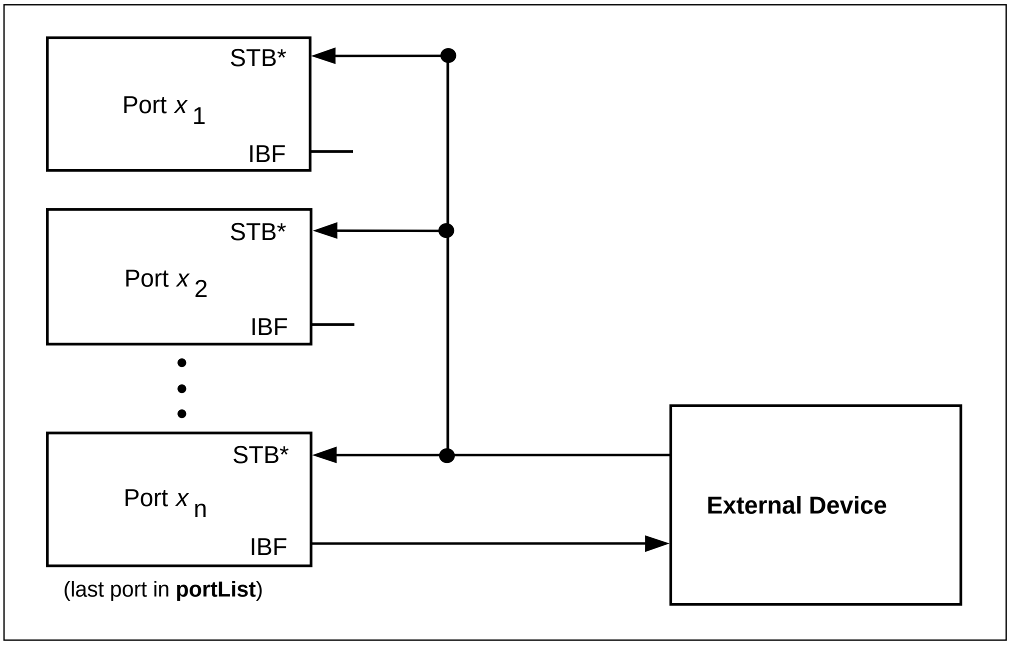


Figure 2-12. Digital Scanning Input Group Handshaking Connections

DIG_SCAN_Setup

Continued

If a group of ports is configured as output, you should not make any connection on the control signals except those for the last port on **portList**. You should make the connection with the external device as if only the last port on **portList** is in the group. No connection is needed for any other port on the list. See Figure 2-13.

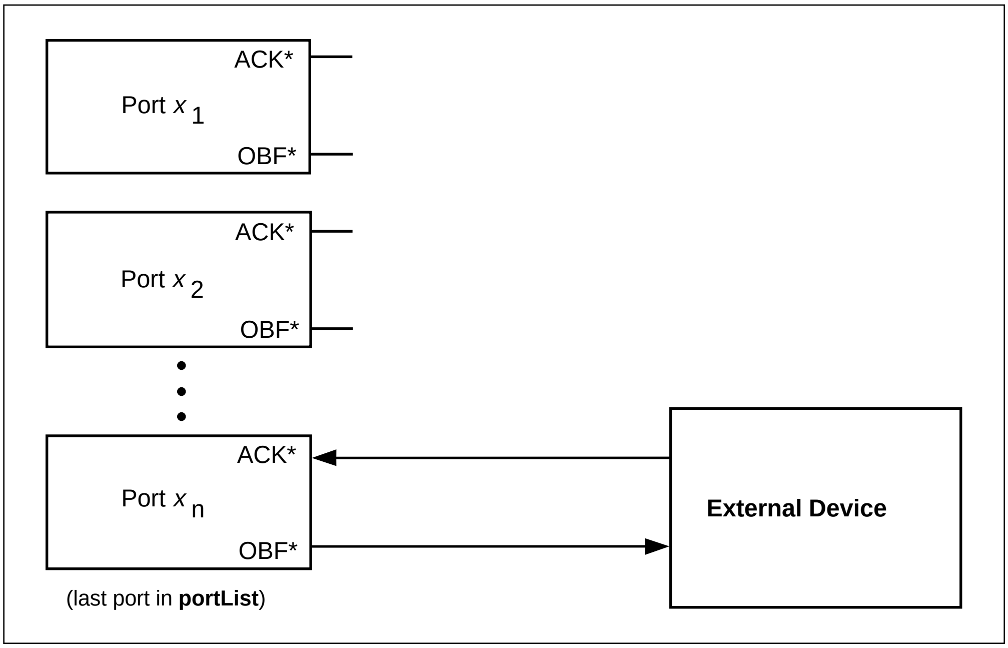


Figure 2-13. Digital Scanning Output Group Handshaking Connections

For DIO-24 users, the correct W1 jumper setting is required to allow `DIG_Block_In` and `DIG_Block_Out` to function properly. If port 0 is configured as a handshaking output port, set jumper W1 to PC4; otherwise, set the jumper to PC6. However, if port 0 is configured as bidirectional, set the jumper to PC2.

Also, if port 0 is configured as bidirectional on a PC-DIO-24, port 1 will not be available.

DSP2200_Calibrate

Format

status = DSP2200_Calibrate (deviceNumber, mode, ADCref)

Purpose

Performs offset calibrations on the analog input and/or analog output of an AT-DSP2200.

Parameters

Input

Name	Type		Description
	Windows	Windows NT	
deviceNumber	I16	I32	assigned by configuration utility
mode	I16	I32	operation to perform
ADCref	I16	I32	reference source for ADC calibration

Parameter Discussion

mode specifies how to calibrate your device.

1: Calibrate analog input.
2: Calibrate analog output.
3: Calibrate both analog input and output.

ADCref specifies the ground reference source used to calibrate the analog input on the AT-DSP2200. NI-DAQ ignores **ADCref** unless the mode is 1 or 3.

0: Use the analog input ground as the reference.
1: Use the external signals connected to ACH0 and ACH1 as ground references.

Using This Function

When the AT-DSP2200 is initialized by `Init_DA_Brds`, NI-DAQ performs an offset calibration on both the analog input and analog output using the analog ground as the reference.

DSP2200_Calibrate

Continued

You can use this function to calibrate the analog input using an external reference or to recalibrate the AT-DSP2200 to compensate for configuration or environmental changes. The external reference specifies the ground voltage level in respect to the signal to be acquired.

DSP2200_Config

Format

status = DSP2200_Config (deviceNumber, aitranslate, aotranslate, demux)

Purpose

Specifies data translation and demultiplexing operations that the AT-DSP2200 can perform on analog input and output data.

Parameters

Input

Name	Type		Description
	Windows	Windows NT	
deviceNumber	I16	I32	assigned by configuration utility
aitranslate	I16	I32	specifies floating-point scaling operations to be performed on analog input
aotranslate	I16	I32	specifies floating-point and scaling operations to be performed on analog output
demux	I16	I32	enables and disables sorting analog input data as it is written to DSP memory

Parameter Discussion

aitranslate is a numeric code that specifies floating-point and scaling operations to be performed on analog input data before NI-DAQ writes the data to memory.

0:	Translation off; data is in 16-bit integer format (default).
1:	Translation on; data is in 32-bit floating-point format.
2:	Scaling on; data is a 32-bit floating-point value in volts.

DSP2200_Config

Continued

aotranslate is a numeric code that specifies floating-point and scaling operations to be performed on analog output data as NI-DAQ reads these data from memory.

0:	Translation off; data is in 16-bit integer format (default).
1:	Translation on; data is in 32-bit floating-point format.
2:	Scaling on; data is a 32-bit floating-point value in volts.

demux enables and disables sorting analog input data as NI-DAQ writes the data to DSP memory. If demux is enabled, NI-DAQ stores data from channel 0 consecutively followed by data from channel 1. If demux is disabled, NI-DAQ stores data from the two channels interleaved in DSP memory.

0:	Demux off (default).
1:	Demux on for acquiring into DSP memory buffer.
2:	Demux on for waveform generation from DSP memory buffer.
3:	Demux on for acquisition and waveform generation when you use DSP memory buffers.

Using This Function

Because NI-DAQ reads data from the ADCs and writes to the DACs through software running locally on the 32C DSP chip, an opportunity exists to manipulate the data on-the-fly. When NI-DAQ writes analog input data to DSP memory, three translation options exist: write unscaled 16-bit integers, write unscaled 32C floating-point numbers, or write scaled 32C floating-point voltages. See the *Allocating DSP Board Memory* section in the *NI-DSP User Manual for PC Compatibles* for information on allocating DSP memory. The **demux** option is only valid when writing analog input data to DSP memory. When **demux** is enabled, NI-DAQ writes data from channel 0 consecutively into DSP memory, beginning at the start of each frame, and channel 1 data is written consecutively beginning at the mid-way point of each frame. When NI-DAQ writes analog input data to PC memory, three similar translation options exist—write unscaled 16-bit integers, write unscaled IEEE single-precision floating-point numbers, or write scaled IEEE single-precision floating-point voltages. Please take care to allocate sufficient memory for analog input data. You need twice as much memory when a floating-point translation option is enabled.

The analog output translation operations are analogous to the input operations except they perform the translations in the opposite direction. If **aotranslate** is 0, the source data must already be in a format suitable for the DACs (16-bit integer DAC values). If **aotranslate** is 1 or 3, the source data are DAC values in 32C format in DSP memory or in IEEE single-precision format in PC memory. If **aotranslate** is 2 or 4, the source data

DSP2200_Config

Continued

are voltages in 32C format in DSP memory or in IEEE single-precision format in PC memory.

Acquisition and waveform generation is most efficient when the buffers are DSP buffers. When you use PC buffers, it is more efficient to turn translation off, because NI-DAQ must transfer twice as much data to the PC if NI-DAQ translates the data to floating-point format. Translation and scaling slightly affect the foreground performance on DSP buffers. When you use PC buffers with the AT-DSP2200, as with most other boards, DMA is more efficient than programmed I/O.

Get_DAQ_Device_Info

Format

status = Get_DAQ_Device_Info (deviceNumber, infoType, infoValue)

Purpose

Allows you to retrieve parameters pertaining to the device operation.

Parameters

Input

Name	Type		Description
	Windows	Windows NT	
deviceNumber	I16	I32	assigned by configuration utility
infoType	U32	U32	type of information you want to retrieve

Output

Name	Type		Description
	Windows	Windows NT	
infoValue	U32	U32	retrieved information

Parameter Discussion

The legal range for the **infoType** is given in terms of constants that are defined in a header file. The header file you should use depends on the language you are using:

- C programmers—`NIDAQCNS.H` (`DATAACQ.H` for LabWindows/CVI)
- BASIC programmers—`NIDAQCNS.INC`
- Pascal programmers—`NIDAQCNS.PAS`

Get_DAQ_Device_Info

Continued

Use **infoType** to let NI-DAQ know which parameter you want to retrieve. **infoValue** will reflect the value of the parameter. **infoValue** will be given either in terms of constants from the header file or as numbers, as appropriate.

infoType can be one of the following:

infoType	**Description**
`ND_BASE_ADDRESS`	Base address, in hexadecimal, of the device specified by **deviceNumber**.

`ND_DATA_XFER_MODE_AI`
`ND_DATA_XFER_MODE_AO_GR1`
`ND_DATA_XFER_MODE_AO_GR2`
`ND_DATA_XFER_MODE_GPCTR0`
`ND_DATA_XFER_MODE_GPCTR1`
`ND_DATA_XFER_MODE_DIO_GR1`
`ND_DATA_XFER_MODE_DIO_GR2`
	See the `Set_DAQ_Device_Info` function for details. `ND_NOT_APPLICABLE` if not relevant to the device.

`ND_DEVICE_TYPE_CODE`	Type of the device specified by **deviceNumber**. See `Init_DA_Brds` for a list of device type codes.

`ND_DMA_A_LEVEL`
`ND_DMA_B_LEVEL`
`ND_DMA_C_LEVEL`
	Level of the DMA channel assigned to the device as channel A, B, and C. `ND_NOT_APPLICABLE` if not relevant or disabled.

`ND_INTERRUPT_A_LEVEL` `ND_INTERRUPT_B_LEVEL`	Level of the interrupt assigned to the device as interrupt A and B. `ND_NOT_APPLICABLE` if not relevant or disabled.
`ND_INTERRUPT_TRIGGER_MODE`	Indicates whether the EISA-A2000 is configured to generate edge-sensitive or level-sensitive interrupts.
`ND_COUNTER_1_SOURCE`	See the `Set_DAQ_Device_Info` function for details. `ND_NOT_APPLICABLE` if not relevant to the device.

☞ **Note:** ***C Programmers*—infoValue *is a pass-by-reference parameter.***

Get_DAQ_Event

Format

status = Get_DAQ_Event (timeOut, handle, message, wParam, lParam)

Purpose

Retrieves a pending event message defined by `Config_DAQ_Event_Message` from the NI-DAQ message queue.

Parameters

Input

Name	**Type**		**Description**
	Windows	**Windows NT**	
timeOut	U32	U32	number of clock ticks to wait for a message

Output

Name	**Type**		**Description**
	Windows	**Windows NT**	
handle	I16	I32	user-defined handle
message	I16	I32	user-defined message
wParam	I16	I32	device and DAQ process status
lParam	I32	U32	number of scans done

Parameter Discussion

timeOut is the number of clock ticks (1 tick is about 55 ms) to wait for a message. There is one special case for **timeout**:

0: Check once and return.

Get_DAQ_Event

Continued

If no message is available and the **timeOut** has elapsed, NI-DAQ returns a **timeOut** status.

If **status** is 0, the value NI-DAQ returns in the **handle** parameter is the same value you passed to `Config_DAQ_Event_Message`, `Config_Alarm_Deadband`, or `Config_ATrig_Event_Message` when you enabled your message. If **status** is not 0, the value NI-DAQ returns in the **handle** parameter is 0.

message is the message you defined passed to `Config_DAQ_Event_Message` if **status** = 0. **message** is 0 if an error has occurred (**status** ≠ 0).

wParam contains two pieces of information if **status** = 0. The lower byte of **wParam** is the device that generated the message and the upper byte of **wParam** is a boolean flag **doneFlag** indicating whether the DAQ process has ended.
doneFlag = 0: Data acquisition is still running.
doneFlag = 1: Data acquisition has stopped.

wParam is 0 if no error has occurred (**status** ≠ 0).

lParam is the number of the scan when NI-DAQ generates the message if **status** = 0. **lParam** is 0 if no error has occurred (**status** ≠ 0).

Using This Function

This function retrieves a message from the NI-DAQ message queue. If no message is in the message queue, the function continuously checks the NI-DAQ message queue until it finds a pending message or timeout value expires. If **timeOut** is 0, this function checks for NI-DAQ event message once and returns.

When `Get_DAQ_Event` retrieves a message from the queue, it removes the message from the queue. To look at the new message without removing it from the queue, call `Peek_DAQ_Event`.

Get_NI_DAQ_Version

Format

status = Get_NI_DAQ_Version (version)

Purpose

Returns the version number of the NI-DAQ library.

Parameter

Output

Name	Type		Description
	Windows	Windows NT	
version	U32	U32	version number assigned

Using This Function

`Get_NI_DAQ_Version` returns a 4-byte value in the **version** parameter. The upper two bytes are reserved and the lower two bytes contain the version number. Always bitwise AND the 4-byte value with FFFF in hex before using the version number. For version 4.9.0, the lower 2-byte value is 490 in hex.

☞ **Note:** ***C Programmers*—version *is a pass-by-reference parameter.***

GPCTR_Change_Parameter

Format

status = GPCTR_Change_Parameter (deviceNumber, gpctrNum, paramID, paramValue)

Purpose

Selects a specific parameter setting for the general-purpose counter (E Series devices only).

☞ **Note:** ***NI-DAQ for Windows NT does not support this function.***

Parameters

Input

Name	Type		Description
	Windows	Windows NT	
deviceNumber	I16	n/a	assigned by configuration utility
gpctrNum	U32	n/a	number of the counter you want to use
paramID	U32	n/a	identification of the parameter you want to change.
paramValue	U32	n/a	new value for the parameter specified by **paramID**

GPCTR_Change_Parameter

Continued

Parameter Discussion

Legal ranges for **gpctrNum**, **paramID**, and **paramValue** are given in terms of constants defined in a header file. The header file you should use depends on the language you are using:

- C programmers—`NIDAQCNS.H` (`DATAACQ.H` for LabWindows/CVI)
- BASIC programmers—`NIDAQCNS.INC` (Visual Basic for Windows programmers should refer to the *Programming Language Considerations* section in Chapter 1 for more information.)
- Pascal programmers—`NIDAQCNS.PAS`

Use **gpctrNum** to indicate to NI-DAQ which counter you want to program. Legal values for this parameter are `ND_COUNTER_0` and `ND_COUNTER_1`.

Legal values for **paramValue** depend on **paramID**. The following paragraphs list legal values for **paramID** with explanations and corresponding legal values for **paramValue**:

paramID = `ND_SOURCE`

The general-purpose counter counts transitions of this signal. Corresponding legal values for **paramValue** are as follows:

- `ND_PFI_0` through `ND_PFI_9`—the 10 I/O connector pins
- `ND_RTSI_0` through `ND_RTSI_6`—the seven RTSI lines
- `ND_INTERNAL_20_MHZ` and `ND_INTERNAL_100_KHZ`—the internal timebases
- `ND_OTHER_GPCTR_TC`—terminal count of the other general-purpose counter

Use this function with **paramID** = `ND_SOURCE_POLARITY` to select polarity of transitions to use for counting.

paramID = `ND_SOURCE_POLARITY`

The general-purpose counter counts the transitions of the signal selected by **paramID** = `ND_SOURCE`. Corresponding legal values for **paramValue** are as follows:

- `ND_LOW_TO_HIGH`—counter counts the low-to-high transitions of the source signal
- `ND_HIGH_TO_LOW`—counter counts the high-to-low transitions of the source signal

GPCTR_Change_Parameter

Continued

paramID = `ND_GATE`

This signal controls the operation of the general-purpose counter in some applications. Corresponding legal values for **paramValue** are as follows:

- `ND_PFI_0` through `ND_PFI_9`—the 10 I/O connector pins
- `ND_RTSI_0` through `ND_RTSI_6`—the seven RTSI lines
- `ND_IN_START_TRIGGER` and `ND_IN_STOP_TRIGGER`—the input section triggers
- `ND_OTHER_GPCTR_OUTPUT`—output of the other general-purpose counter

Use this function with **paramID** = `ND_GATE_POLARITY` to select polarity of the gate signal.

paramID = `ND_GATE_POLARITY`

This gate signal controls the operation of the general-purpose counter in some applications. In those applications, you can use polarity of the gate signals to modify behavior of the counter. Corresponding legal values for **paramValue** are as follows:

- `ND_POSITIVE`
- `ND_NEGATIVE`

The meaning of the two `ND_GATE_POLARITY` selections is described in the `GPCTR_Set_Application` function.

paramID = `ND_INITIAL_COUNT`

The general-purpose counter starts counting from this number when the counter is configured for one of the simple event counting and time measurement applications. Corresponding legal values for **paramValue** are 0 through 2^{24}-1.

paramID = `ND_COUNT_1, ND_COUNT_2, ND_COUNT_3, ND_COUNT_4`

The general-purpose counter uses these numbers for pulse width specification when the counter is configured for one of the simple pulse and pulse train generation applications. For example, when you use the counter for frequency shift keying (FSK), `ND_COUNT_1` and `ND_COUNT_2` specify the durations of low and high output states for one gate state and `ND_COUNT_3` and `ND_COUNT_4` specify them for the other gate state. Corresponding legal values for **paramValue** are 2 through 2^{24}-1.

GPCTR_Change_Parameter

Continued

paramID = `ND_AUTOINCREMENT_COUNT`

The value specified by `ND_COUNT_1` is incremented by the value selected by `ND_AUTOINCREMENT_COUNT` every time the counter is reloaded with the value specified by `ND_COUNT_1`.

For example, with this feature you can generate retriggerable delayed pulses with incrementally increasing delays. You can then use these pulses for applications such as equivalent time sampling (ETS). Corresponding legal values for **paramValue** are 0 through 2^8-1.

paramID = `ND_UP_DOWN`

When the application is `ND_SIMPLE_EVENT_CNT` or `ND_BUFFERED_EVENT_CNT`, you can use the up or down control options of the DAQ-STC general purpose counters. The up or down control can be performed by software or hardware.

Software Control

The software up or down control is available by default; if you do not use the `GPCTR_Change_Parameter` function with **paramID** set to `ND_UP_DOWN`, the counter will be configured for the software up or down control and will start counting up. To make the counter use the software up or down control and start counting down, use the `GPCTR_Change_Parameter` function with the **paramID** set to `ND_UP_DOWN` and the **paramValue** set to `ND_COUNT_DOWN`. To change the counting direction during counting, use the `GPCTR_Control` function with the action set to `ND_COUNT_UP` or `ND_COUNT_DOWN`.

Hardware Control

If you want to use hardware to control the counting direction, use digital I/O line 6 (GPCTR 0) or 7 (GPCTR 1); the counter will count down when the DIO line is in the low state and up when it is in the high state. Use the `GPCTR_Change_Parameter` function with the **paramID** set to `ND_UP_DOWN` and the **paramValue** set to `ND_HARDWARE` to take advantage of this counter feature.

paramID = `ND_OUTPUT_MODE`

This value changes the output mode from default toggle (the output of the counter toggles on each terminal count) to pulsed (the output of the counter makes a pulse on each terminal count). The corresponding settings of **paramValue** are `ND_PULSE` and `ND_TOGGLE`.

GPCTR_Change_Parameter

Continued

paramID = `ND_OUTPUT_POLARITY`

This **paramID** allows you to change the output polarity from default positive (the normal state of the output is TTL low) to negative (the normal state of the output is TTL high). The corresponding settings of **paramValue** are `ND_POSITIVE` and `ND_NEGATIVE`.

Using This Function

This function lets you customize the counter for your application. You can use this function after the `GPCTR_Set_Application` function, and before `GPCTR_Control` function with **action** = `ND_PREPARE` or **action** = `ND_PROGRAM`. You can call this function as many times as you need to.

GPCTR_Config_Buffer

Format

status = GPCTR_Config_Buffer (deviceNumber, gpctrNum, reserved, numPoints, buffer)

Purpose

Assigns a buffer that NI-DAQ will use for a buffered counter operation (E Series devices only).

☞ **Note:** ***NI-DAQ for Windows NT does not support this function.***

Parameters

Input

Name	Type		Description
	Windows	Windows NT	
deviceNumber	I16	n/a	assigned by configuration utility
gpctrNum	U32	n/a	number of the counter you want to use
reserved	U32	n/a	reserved parameter, must be 0
numPoints	U32	n/a	number of data points the buffer can hold
buffer	[U32], HDL	n/a	used to hold counts

Parameter Discussion

The legal range for **gpctrNum** is given in terms of constants defined in a header file. The header file you should use depends on the language you are using:

- C programmers—`NIDAQCNS.H` (`DATAACQ.H` for LabWindows/CVI)
- BASIC programmers—`NIDAQCNS.INC` (Visual Basic for Windows programmers should refer to the *Programming Language Considerations* section in Chapter 1 for more information.)
- Pascal programmers—`NIDAQCNS.PAS`

GPCTR_Config_Buffer

Continued

Use **gpctrNum** to indicate to NI-DAQ which counter you want to program. Legal values for this parameter are `ND_COUNTER_0` and `ND_COUNTER_1`.

numPoints is the number of data points the **buffer** can hold. The definition of a data point depends on the application the counter is used for. Legal range is 2 through 2^{32}-1.

When you use the counter for one of the buffered event counting or buffered time measurement operations, a data point is a single counted number.

buffer is an array of U32 or an `NI_DAQ_Mem` array.

Using This Function

You need to use this function if you want to use a general-purpose counter for buffered operation. You should call this function after calling the `GPCTR_Set_Application` function.

NI-DAQ transfers counted values into the **buffer** assigned by this function when you are performing a buffered counter operation.

If you are using the general-purpose counter for `ND_BUFFERED_PERIOD_MSR`, `ND_BUFFERED_SEMI_PERIOD_MSR`, or `ND_BUFFERED_PULSE_WIDTH_MSR`, you should wait for the operation to be completed before accessing the buffer.

GPCTR_Control

Format

status = GPCTR_Control (deviceNumber, gpctrNum, action)

Purpose

Controls the operation of the general-purpose counter (E Series devices only).

☞ **Note:** ***NI-DAQ for Windows NT does not support this function.***

Parameters

Input

Name	Type		Description
	Windows	Windows NT	
deviceNumber	I16	n/a	assigned by configuration utility
gpctrNum	U32	n/a	number of the counter you want to use
action	U32	n/a	the action you want NI-DAQ to take

GPCTR_Control

Continued

Parameter Discussion

Legal ranges for the **gpctrNum** and **action** are given in terms of constants defined in a header file. The header file you should use depends on the language you are using:

- C programmers—`NIDAQCNS.H` (`DATAAQC.H` for LabWindows/CVI)
- BASIC programmers—`NIDAQCNS.INC` (Visual Basic for Windows programmers should refer to the *Programming Language Considerations* section in Chapter 1 for more information.)
- Pascal programmers—`NIDAQCNS.PAS`

Use **gpctrNum** to indicate to NI-DAQ which counter you want to program. Legal values for this parameter are `ND_COUNTER_0` and `ND_COUNTER_1`.

action is what you want NI-DAQ to perform with the counter. Legal values for this parameter are as follows:

Action	**Description**
`ND_PREPARE`	Prepare the general-purpose counter for the operation selected by invocations of the `GPCTR_Set_Application` and (optionally) `GPCTR_Change_Parameter` function. Do not arm the counter.
`ND_ARM`	Arm the general-purpose counter.
`ND_PROGRAM`	`ND_PREPARE` and then `ND_ARM` the counter.
`ND_RESET`	Reset the general-purpose counter.
`ND_COUNT_UP`	Change the counting direction to UP. Please see *Using This Function*.
`ND_COUNT_DOWN`	Change the counting direction to DOWN. Please see *Using This Function*.

GPCTR_Control

Continued

Using This Function

You need to use this function with **action** = `PROGRAM` after completing the configuration sequence consisting of calling `GPCTR_Set_Application` followed by optional calls to `GPCTR_Change_Parameter` and `GPCTR_Config_Buffer`.

Use the `ND_PREPARE` and `ND_ARM` actions to program the counter before arming. You may find this useful if it is critical to minimize time between a software event (a call to `GPCTR_Control`) and a hardware action (counter starts counting).

You can use this function with **action** = `ND_RESET` whenever you want to halt the operation the general-purpose counter is performing.

Use actions `ND_COUNT_UP` and `ND_COUNT_DOWN` to change the counting direction. You can do this only when your application is `ND_SIMPLE_EVENT_CNT` or `ND_BUFFERED_EVENT_CNT` and the counter is configured for software control of the counting direction (up/down).

GPCTR_Set_Application

Format

status = GPCTR_Set_Application (deviceNumber, gpctrNum, application)

Purpose

Selects the application for which you will use the general-purpose counter (E Series devices only).

☞ **Note:** ***NI-DAQ for Windows NT does not support this function.***

Parameters

Input

Name	Type		Description
	Windows	Windows NT	
deviceNumber	I16	n/a	assigned by configuration utility
gpctrNum	U32	n/a	number of the counter you want to use
application	U32	n/a	application for which you want to use the counter

Parameter Discussion

Legal ranges for **gpctrNum** and **application** are given in terms of constants that are defined in a header file. The header file you should use depends on the language you are using:

- C programmers—`NIDAQCNS.H` (`DATAACQ.H` for LabWindows/CVI)
- BASIC programmers—`NIDAQCNS.INC` (Visual Basic for Windows programmers should refer to the *Programming Language Considerations* section in Chapter 1 for more information.)
- Pascal programmers—`NIDAQCNS.PAS`

GPCTR_Set_Application

Continued

Use **gpctrNum** to indicate to NI-DAQ which counter you want to program. Legal values for this parameter are `ND_COUNTER_0` and `ND_COUNTER_1`.

application can be one of the following:

Group	Application	Description
Simple Counting and Time Measurement	`ND_SIMPLE_EVENT_CNT`	Simple event counting
	`ND_SINGLE_PERIOD_MSR`	Simple single period measurement
	`ND_SINGLE_PULSE_WIDTH_MSR`	Simple single pulse-width measurement
	`ND_TRIG_PULSE_WIDTH_MSR`	Pulse-width measurement you can use for recurring pulses.
Simple Pulse and Pulse Train Generation	`ND_SINGLE_PULSE_GNR`	Generation of a single pulse
	`ND_SINGLE_TRIG_PULSE_GNR`	Generation of a single triggered pulse
	`ND_RETRIG_PULSE_GNR`	Generation of a retriggerable single pulse
	`ND_PULSE_TRAIN_GNR`	Generation of pulse train
	`ND_FSK`	Frequency Shift-Keying
Buffered Counting and Time Measurement	`ND_BUFFERED_EVENT_CNT`	Buffered, asynchronous event counting
	`ND_BUFFERED_PERIOD_MSR`	Buffered, asynchronous period measurement
	`ND_BUFFERED_SEMI_PERIOD_MSR`	Buffered, asynchronous semi-period measurement
	`ND_BUFFERED_PULSE_WIDTH_MSR`	Buffered, asynchronous pulse-width measurement
`_CNT` stands for *Counting* `_MSR` stands for *Measurement* `_GNR` stands for *Generation*		

GPCTR_Set_Application

Continued

Using This Function

NI-DAQ requires you to select a set of parameters so that it can program the counter hardware. Those parameters include, for example, signals to be used as counter source and gate and the polarities of those signals. A full list of the parameters is given in the description of the `GPCTR_Change_Parameter` function. By using the `GPCTR_Set_Application` function, you assign specific values to all of those parameters. If you do not like some of the settings used by this function, you can alter them by using the `GPCTR_Change_Parameter` function.

When using DMA for buffered `GPCTR` operations on E Series devices, we recommend you use the internal 20 MHz timebase over the internal 100 kHz timebase. The 100 kHz timebase will not work correctly when you are using DMA. For measuring gate signals slower than the internal 20 MHz timebase will allow, when you need to use DMA, we recommend using external timebases. We can use DMA operations on typical 486-based machines without any errors for gate signals of up to 50 kHz using the internal 20 MHz timebase. Trying to achieve rates higher than 50 kHz may cause **gpctrDataLossError**. This error may cause some computers to lock up because of a memory parity error.

The behavior of the counter you are preparing for an application with this function will depend on **application**, your future calls of the `GPCTR` functions, and the signals supplied to the counter. The following paragraphs illustrate typical scenarios.

application = `ND_SIMPLE_EVENT_CNT`

In this application, the counter is used for simple counting of events. By default, the events are low-to-high transitions on the PFI8/GPCTR0_SOURCE I/O connector pin for general-purpose counter 0 and the PFI3/GPCTR1_SOURCE I/O connector pin for general-purpose counter 1. The counter counts up starting from 0, and it is not gated.

Figure 2-14 shows one possible scenario of a counter used for `ND_SIMPLE_EVENT_CNT` after the following programming sequence:

```
GPCTR_Control(deviceNumber, gpctrNum, ND_RESET)
GPCTR_Set_Application(deviceNumber, gpctrNum, ND_SIMPLE_EVENT_CNT)
GPCTR_Control(deviceNumber, gpctrNum, ND_PROGRAM)
```

GPCTR_Set_Application

Continued

In Figure 2-14:

- Source is the signal present at the counter source input
- Count is the value you would read from the counter if you called the `GPCTR_Watch` function with **entityID** = `ND_COUNT`. The different numbers illustrate behavior at different times.

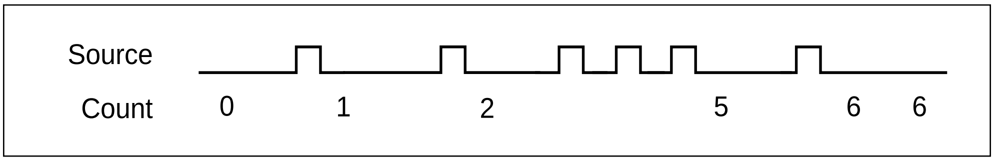


Figure 2-14. Simple Event Counting

The following pseudo-code continuation of the example given earlier illustrates what you can do if you want to continuously read the counter value (`GPCTR_Watch` function with **entityID** = `ND_COUNT` does this) and print it:

```
Repeat Forever
{
GPCTR_Watch(deviceNumber, gpctrNum, COUNT, counterValue)
Output counterValue.
}
```

When the counter reaches 2^{24}-1 (Terminal Count) it rolls over and keeps counting. If you want to check if this occurred, use `GPCTR_Watch` function with **entityID** set to `ND_TC_REACHED`.

Typically, you will find modifying the following parameters through the `GPCTR_Change_Parameter` function useful when the counter **application** is `ND_SIMPLE_EVENT_CNT`. You can change the following:

- `ND_SOURCE` to any value
- `ND_SOURCE_POLARITY` to `ND_HIGH_TO_LOW`

You can use the `GPCTR_Change_Parameter` function after calling `GPCTR_Set_Application` and before calling `GPCTR_Control` with **action** = `ND_PROGRAM` or `ND_PREPARE`.

GPCTR_Set_Application

Continued

application = `ND_SINGLE_PERIOD_MSR`

In this application, the counter is used for a single measurement of the time interval between two transitions of the same polarity of the gate signal. By default, the events are low-to-high transitions on the PFI9/GPCTR0_GATE I/O connector pin for general-purpose counter 0 and the PFI4/GPCTR1_GATE I/O connector pin for general-purpose counter 1. The counter counts the 20 MHz internal timebase (`ND_INTERNAL_20_MHZ`), so the resolution of measurement is 50 ns. The counter counts up starting from 0.

With the default 20 MHz timebase, combined with the counter width (24 bits), you can measure a time interval between 100 ns and 0.8 s long.

Figure 2-15 shows one possible scenario of a counter used for `ND_SINGLE_PERIOD_MSR` after the following programming sequence:

```
GPCTR_Control(deviceNumber, gpctrNum, ND_RESET)

GPCTR_Set_Application(deviceNumber, gpctrNum, ND_SINGLE_PERIOD_MSR)

GPCTR_Control(deviceNumber, gpctrNum, ND_PROGRAM)
```

In Figure 2-15:

- Gate is the signal present at the counter gate input.
- Source is the signal present at the counter source input.
- Count is the value you would read from the counter if you called the `GPCTR_Watch` function with **entityID** = `ND_COUNT`. The different numbers illustrate behavior at different times.
- Armed is the value you would read from the counter if you called the `GPCTR_Watch` function with **entityID** = `ND_ARMED`. The different values illustrate behavior at different times.

GPCTR_Set_Application

Continued

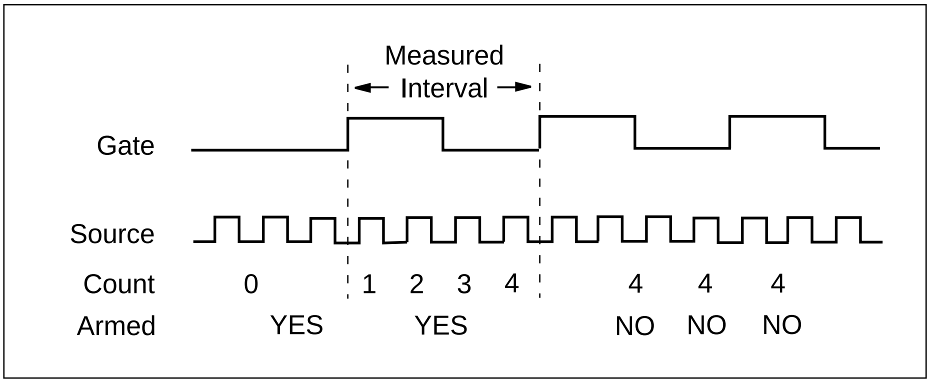


Figure 2-15. Single Period Measurement

Use the `GPCTR_Watch` function with **entityID** = `ND_ARMED` to monitor the progress of the counting process. This measurement completes when **entityValue** becomes `ND_NO`. When counter is no longer armed, you can retrieve the counted value by using `GPCTR_Watch` with **entityID** = `ND_COUNT`. You can do this as follows:

```
Create U32 variable counter_armed.
Create U32 variable counted_value.
repeat
{
      GPCTR_Watch(deviceNumber, gpctrNumber, ND_ARMED, counter_armed)
}
until (counter_armed = ND_NO)
GPCTR_Watch(deviceNumber, gpctrNumber, ND_COUNT, counted_value)
```

To calculate the measured interval, you need to multiply the counted value by the period corresponding to the timebase you are using. For example, if your `ND_SOURCE` is `ND_INTERNAL_20_MHZ`, the interval will be

1/(20 MHz) = 50 ns. If the `ND_COUNT` is 4 (Figure 2-15), the actual interval is 4 * 50 ns = 200 ns.

GPCTR_Set_Application

Continued

When the counter reaches 2^{24}-1 (Terminal Count), it rolls over and keeps counting. If you want to check if this occurred, use the `GPCTR_Watch` function with **entityID** set to `ND_TC_REACHED`.

Typically, you will find modifying the following parameters through the `GPCTR_Change_Parameter` function useful when the counter **application** is `ND_SINGLE_PERIOD_MSR`. You can change the following:

- `ND_SOURCE` to `ND_INTERNAL_100_KHZ`. With this timebase, you can measure the time interval between 20 µs and 160 s. The resolution will be lower than if you are using the `ND_INTERNAL_20_MHZ` timebase.
- `ND_SOURCE_POLARITY` to `ND_HIGH_TO_LOW`.
- `ND_GATE` to any legal value listed in the `GPCTR_Change_Parameter` function description.
- `ND_GATE_POLARITY` to `ND_NEGATIVE`. The interval will be measured from a high-to-low to the next high-to-low transition of the gate signal.

You can use the `GPCTR_Change_Parameter` function after calling `GPCTR_Set_Application` and before calling `GPCTR_Control` with **action** = `ND_PROGRAM` or `ND_PREPARE`.

If you want to provide your timebase, you can connect your timebase source to one of the PFI pins on the I/O connector and change `ND_SOURCE` and `ND_SOURCE_POLARITY` to the appropriate values.

You can also configure the other general-purpose counter for `ND_PULSE_TRAIN_GNR` and set `ND_SOURCE` of this counter to `ND_OTHER_GPCTR_TC` if you want to measure intervals longer than 160 s.

application = `ND_SINGLE_PULSE_WIDTH_MSR`

In this application, the counter is used for a single measurement of the time interval between two transitions of the opposite polarity of the gate signal. By default, the measurement is performed between a low-to-high and a high-to-low transition on the PFI9/GPCTR0_GATE I/O connector pin for general-purpose counter 0 and the PFI4/GPCTR1_GATE I/O connector pin for general-purpose counter 1. The counter counts the 20 MHz internal timebase (INTERNAL_20_MHZ), so the resolution of measurement is 50 ns. The counter counts up starting from 0.

The default 20 MHz timebase, combined with the counter width (24 bits), lets you measure the duration of a pulse between 100 ns and 0.8 s long.

GPCTR_Set_Application

Continued

Figure 2-16 shows one possible scenario of a counter used for `ND_SINGLE_PULSE_WIDTH_MSR` after the following programming sequence:

```
GPCTR_Control(deviceNumber, gpctrNum, ND_RESET)
GPCTR_Set_Application(deviceNumber, gpctrNum, ND_SINGLE_PULSE_WIDTH_MSR)
GPCTR_Control(deviceNumber, gpctrNum, ND_PROGRAM)
```

In Figure 2-16:

- Gate is the signal present at the counter gate input.
- Source is the signal present at the counter source input.
- Count is the value you would read from the counter if you called the `GPCTR_Watch` function with **entityID** = `ND_COUNT`. The different numbers illustrate behavior at different times.
- Armed is the value you would read from the counter if you called the `GPCTR_Watch` function with **entityID** = `ND_ARMED`. The different values illustrate behavior at different times.

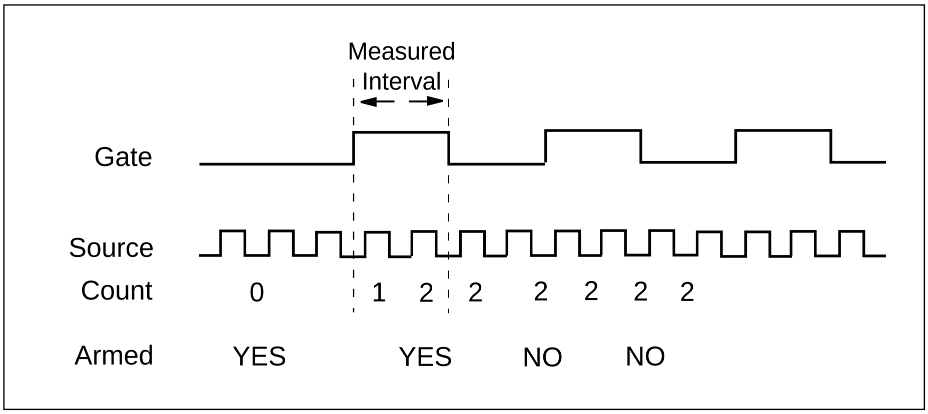


Figure 2-16. Single Pulse Width Measurement

Use the `GPCTR_Watch` function with **entityID** = `ND_ARMED` to monitor the progress of the counting process. This measurement completes when **entityValue** becomes `ND_NO`. When the counter is no longer armed, you can retrieve the counted value by

GPCTR_Set_Application

Continued

using `GPCTR_Watch` with **entityID** = `ND_COUNT`, as shown in the following example code:

```
Create U32 variable counter_armed.

Create U32 variable counted_value.

repeat

{

      GPCTR_Watch(deviceNumber, gpctrNumber, ND_ARMED, counter_armed)

}

until (counter_armed = ND_NO)

GPCTR_Watch(deviceNumber, gpctrNumber, ND_COUNT, counted_value)
```

To calculate the measured interval, multiply the counted value by the period corresponding to the timebase you are using. For example, if your `ND_SOURCE` is `ND_INTERNAL_20_MHZ`, the interval will be 1/(20 MHz) = 50 ns. If the `ND_COUNT` is 4 (Figure 2-15), the actual interval is 4 * 50 ns = 200 ns.

When the counter reaches 2^{24}-1 (Terminal Count), it rolls over and keeps counting. If you want to check if this occurred, use the `GPCTR_Watch` function with **entityID** set to `ND_TC_REACHED`.

Typically, you will find modifying the following parameters through the `GPCTR_Change_Parameter` function useful when the counter **application** is `ND_SINGLE_PULSE_WIDTH_MSR`. You can change the following:

- `ND_SOURCE` to `ND_INTERNAL_100_KHZ`. With this timebase, you can measure pulse widths between 20 µs and 160 s. The resolution will be lower than if you are using the `ND_INTERNAL_20_MHZ` timebase.
- `ND_SOURCE_POLARITY` to `ND_HIGH_TO_LOW`.
- `ND_GATE` to any legal value listed in the `GPCTR_Change_Parameter` function description.
- `ND_GATE_POLARITY` to `ND_NEGATIVE`. The pulse width will be measured from a high-to-low to the next low-to-high transition of the gate signal.

You can use the `GPCTR_Change_Parameter` function after calling `GPCTR_Set_Application` and before calling `GPCTR_Control` with **action** = `ND_PROGRAM` or `ND_PREPARE`.

GPCTR_Set_Application

Continued

If you want to provide your timebase, connect your timebase source to one of the PFI pins on the I/O connector and change `ND_SOURCE` and `ND_SOURCE_POLARITY` to the appropriate values.

You can also configure the other general-purpose counter for `ND_PULSE_TRAIN_GNR` and set `ND_SOURCE` of this counter to `ND_OTHER_GPCTR_TC` if you want to measure pulse widths longer than 160 s.

Warning: ***Application*** `ND_SINGLE_PULSE_WIDTH_MSR` ***will work as described only if the gate signal stays in the low state when*** `ND_GATE_POLARITY` ***is*** `ND_POSITIVE`***, or if the signal stays in the high state when*** `ND_GATE_POLARITY` ***is*** `ND_NEGATIVE` ***while*** `GPCTR_Control` ***is executed with*** **action** = `ND_ARM` ***or*** **action** = `ND_PROGRAM`***. If this criterion is not met, executing*** `GPCTR_Control` ***with action*** = `ND_ARM` ***or*** **action** = `ND_PROGRAM` ***returns*** **gateSignalError.** ***If this happens, you should not rely on values returned by*** `GPCTR_Watch`**.**

application = `ND_TRIG_PULSE_WIDTH_MSR`

In this application, the counter is used for a single measurement of the time interval between two transitions of the opposite polarity of the gate signal. By default, the measurement is performed between a low-to-high and a high-to-low transition on the PFI9/GPCTR0_GATE I/O connector pin for general-purpose counter 0 and the PFI4/GPCTR1_GATE I/O connector pin for general-purpose counter 1. The counter counts the 20 MHz internal timebase (INTERNAL_20_MHZ), so the resolution of measurement is 50 ns. The counter counts up starting from 0.

Unlike `ND_SINGLE_PULSE_WIDTH_MSR`, your gate signal can change state during counter arming. However, the counter will start counting only after a high-to-low edge on the gate if the gate polarity is positive, or after a low-to-high edge on the gate if the gate polarity is negative. This transition is the trigger from this application's name.

The default 20 MHz timebase, combined with the counter width (24 bits), lets you measure the duration of a pulse between 100 ns and 0.8 s long.

Figure 2-17 shows one possible scenario of a counter used for `ND_TRIG_PULSE_WIDTH_MSR` after the following programming sequence:

```
GPCTR_Control(deviceNumber, gpctrNum, ND_RESET)

GPCTR_Set_Application(deviceNumber, gpctrNum, ND_TRIG_PULSE_WIDTH_MSR)

GPCTR_Control(deviceNumber, gpctrNum, ND_PROGRAM)
```

GPCTR_Set_Application

Continued

In Figure 2-17:

- Gate is the signal present at the counter gate input.
- Source is the signal present at the counter source input.
- Count is the value you would read from the counter if you called the `GPCTR_Watch` function with **entityID** = `ND_COUNT`. The different numbers illustrate behavior at different times.
- Armed is the value you would read from the counter if you called the `GPCTR_Watch` function with **entityID** = `ND_ARMED`. The different values illustrate behavior at different times.

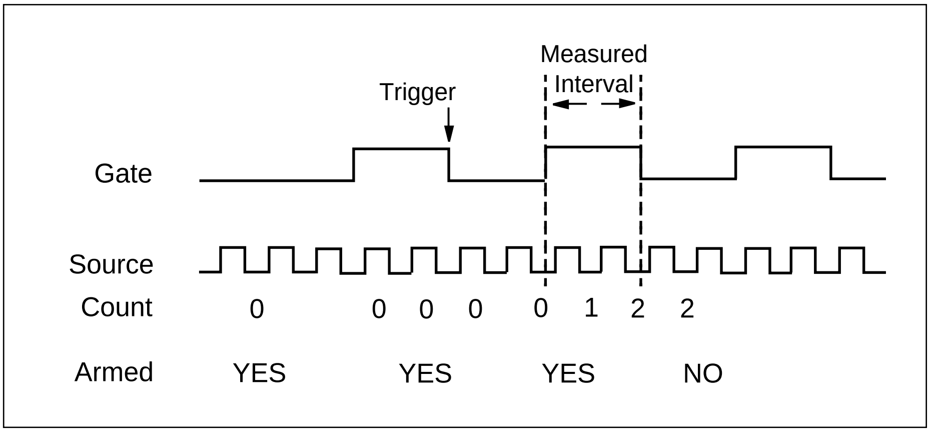


Figure 2-17. Single Triggered Pulse Width Measurement

Use the `GPCTR_Watch` function with **entityID** = `ND_ARMED` to monitor the progress of the counting process. This measurement completes when **entityValue** becomes `ND_NO`. When the counter is no longer armed, you can retrieve the counted value by using `GPCTR_Watch` with **entityID** = `ND_COUNT`, as shown in the following example code:

```
Create U32 variable counter_armed.

Create U32 variable counted_value.

repeat

{

        GPCTR_Watch(deviceNumber, gpctrNumber, ND_ARMED, counter_armed)
```

GPCTR_Set_Application

Continued

```
}
until (counter_armed = ND_NO)


GPCTR_Watch(deviceNumber, gpctrNumber, ND_COUNT, counted_value)
```

To calculate the measured interval, multiply the counted value by the period corresponding to the timebase you are using. For example, if your `ND_SOURCE` is `ND_INTERNAL_20_MHZ`, the interval will be 1/(20 MHz) = 50 ns. If the `ND_COUNT` is 4 (Figure 2-15), the actual interval is 4 * 50 ns = 200 ns.

When the counter reaches 2^{24}-1 (Terminal Count), it rolls over and keeps counting. If you want to check if this occurred, use `GPCTR_Watch` function with **entityID** set to `ND_TC_REACHED`.

Typically, you will find modifying the following parameters through the `GPCTR_Change_Parameter` function useful when the counter **application** is `ND_TRIG_PULSE_WIDTH_MSR`. You can change the following:

- `ND_SOURCE` to `ND_INTERNAL_100_KHZ`. With this timebase, you can measure pulse widths between 20 μs and 160 s. The resolution will be lower than if you are using the `ND_INTERNAL_20_MHZ` timebase.
- `ND_SOURCE_POLARITY` to `ND_HIGH_TO_LOW`.
- `ND_GATE` to any legal value listed in the `GPCTR_Change_Parameter` function description.
- `ND_GATE_POLARITY` to `ND_NEGATIVE`. The pulse width will be measured from a high-to-low to the next low-to-high transition of the gate signal.

You can use the `GPCTR_Change_Parameter` function after calling `GPCTR_Set_Application` and before calling `GPCTR_Control` with **action** = `ND_PROGRAM` or `ND_PREPARE`.

If you want to provide your timebase, connect your timebase source to one of the PFI pins on the I/O connector and change `ND_SOURCE` and `ND_SOURCE_POLARITY` to the appropriate values.

You can also configure the other general-purpose counter for `ND_PULSE_TRAIN_GNR` and set `ND_SOURCE` of this counter to `ND_OTHER_GPCTR_TC` if you want to measure pulse widths longer than 160 s.

GPCTR_Set_Application

Continued

application = `ND_SINGLE_PULSE_GNR`

In this application, the counter is used for the generation of single delayed pulse. By default, you get this through the 20 MHz internal timebase (`ND_INTERNAL_20_MHZ`), so the resolution of timing is 50 ns. By default, the counter counts down from `ND_COUNT_1` = 5 million to 0 for the delay time and then down from `ND_COUNT_2` = 10 million to 0 for the pulse generation time to generate a 0.5 s pulse after 0.25 s of delay.

The default 20 MHz timebase, combined with the counter width (24 bits), lets you generate pulses with a delay and length between 100 ns and 0.8 s (each).

For example, assume that you want to generate a pulse 200 ns long after 150 ns of delay. You need to set `ND_COUNT_1` to 150 ns/50 ns = 3 and `ND_COUNT_2` to 200 ns/50 ns = 4. Figure 2-18 shows a counter used for `ND_SINGLE_PULSE_GNR` after the following programming sequence:

```
GPCTR_Control(deviceNumber, gpctrNum, ND_RESET)

GPCTR_Set_Application(deviceNumber, gpctrNum,
ND_SINGLE_PULSE_GRN)

GPCTR_Change_Parameter(deviceNumber, gpctrNum, ND_COUNT_1, 3)

GPCTR_Change_Parameter(deviceNumber, gpctrNum, ND_COUNT_2, 4)

Select_Signal(deviceNumber, gpctrNumOut,
gpctrNumOut,ND_LOW_TO_HIGH)

GPCTR_Control(deviceNumber, gpctrNum, ND_PROGRAM)
```

In Figure 2-18:

- Source is the signal present at the counter source input.
- Output is the signal present at the counter output.
- Armed is the value you would read from the counter if you called the `GPCTR_Watch` function with **entityID** = `ND_ARMED`. The different values illustrate behavior at different times.

GPCTR_Set_Application

Continued

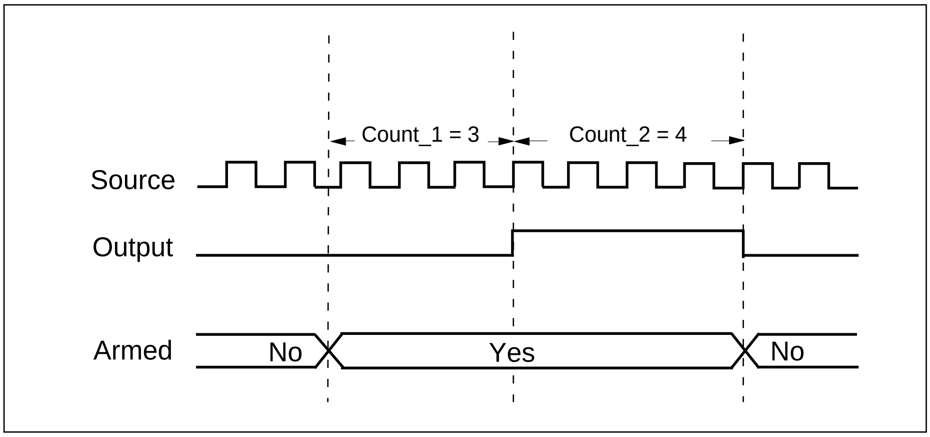


Figure 2-18. Single Pulse Generation

Use the GPCTR_Watch function with **entityID** = ND_ARMED to monitor the progress of the pulse generation process. The generation completes when **entityValue** becomes ND_NO.

You will typically find modification of the following parameters through the GPCTR_Change_Parameter function useful when the counter **application** is ND_SINGLE_PERIOD_MSR. You can change the following:

- ND_COUNT_1 and ND_COUNT_2 to any value between 2 and 2^{24} - 1. The defaults are given for illustrative purposes only.
- ND_SOURCE to ND_INTERNAL_100_KHZ. With this timebase, you can generate pulses with delay and length between 20 µs and 160 s. The timing resolution will be lower than if you are using the ND_INTERNAL_20_MHZ timebase.

You can use the GPCTR_Change_Parameter function after calling GPCTR_Set_Application and before calling GPCTR_Control with **action** = ND_PROGRAM or ND_PREPARE.

If you want to provide your timebase, connect your timebase source to one of the PFI pins on the I/O connector and change ND_SOURCE and ND_SOURCE_POLARITY to the appropriate values.

GPCTR_Set_Application

Continued

You can also configure the other general-purpose counter for `ND_PULSE_TRAIN_GNR` and set `ND_SOURCE` of this counter to `ND_OTHER_GPCTR_TC` if you want to generate pulses with delays and intervals longer than 160 s.

application = `ND_SINGLE_TRIG_PULSE_GRN`

In this application, the counter is used for the generation of single delayed pulse after a transition on the gate input. By default, this is achieved by using the 20 MHz internal timebase (`ND_INTERNAL_20_MHZ`), so the resolution of timing is 50 ns. By default, the counter counts down from `ND_COUNT_1` = 5 million to 0 for the delay time, and then down from `ND_COUNT_2` = 10 million to 0 for the pulse generation time to generate a 0.5 s pulse after 0.25 s of delay. By default, the gate is PFI9/GPCTR0_GATE I/O connector pin for general-purpose counter 0 and the PFI4/GPCTR1_GATE I/O connector pin for general-purpose counter 1, and the transition which initiates the pulse generation is low-to-high. Only the first transition of the gate signal after you arm the counter initiates pulse generation; all subsequent transitions are ignored.

The default 20 MHz timebase, combined with the counter width (24 bits), lets you generate pulses with delay and length between 100 ns and 0.8 s.

Assume that you want to generate a pulse 200 ns long after 150 ns of delay from the transition of the gate signal. You need to set `ND_COUNT_1` to 150 ns/50 ns = 3 and `ND_COUNT_2` to 200 ns/50 ns = 4. Figure 2-19 shows the scenario of a counter used for `ND_SINGLE_TRIG_PULSE_GRN` after the following programming sequence:

```
GPCTR_Control(deviceNumber, gpctrNum, ND_RESET)

GPCTR_Set_Application(deviceNumber, gpctrNum, ND_SINGLE_TRIG_PULSE_GRN)

GPCTR_Change_Parameter(deviceNumber, gpctrNum, ND_COUNT_1, 3)

GPCTR_Change_Parameter(deviceNumber, gpctrNum, ND_COUNT_2, 4)

Select_Signal(deviceNumber, gpctrNumOut, gpctrNumOut,ND_LOW_TO_HIGH)

GPCTR_Control(deviceNumber, gpctrNum, ND_PROGRAM)
```

In Figure 2-19:

- Gate is the signal present at the counter gate input.
- Source is the signal present at the counter source input.
- Output is the signal present at the counter output.

GPCTR_Set_Application

Continued

- Armed is the value you would read from the counter if you called the `GPCTR_Watch` function with **entityID** = `ND_ARMED`. The different values illustrate behavior at different times.

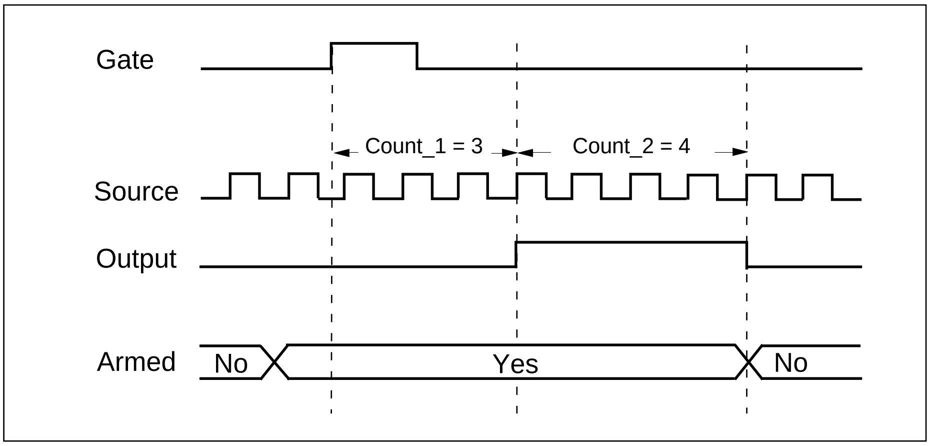


Figure 2-19. Single Triggered Pulse Generation

Use the `GPCTR_Watch` function with **entityID** = `ND_ARMED` to monitor the progress of the pulse generation process. The generation completes when **entityValue** becomes `ND_NO`.

Typically, you will find modifying the following parameters through the `GPCTR_Change_Parameter` function useful when the counter **application** is `ND_SINGLE_TRIG_PULSE_GNR`. You can change the following:

- `ND_COUNT_1` and `ND_COUNT_2` to any value between 2 and 2^{24} - 1. The defaults are given for illustrative purposes only.
- `ND_SOURCE` to `ND_INTERNAL_100_KHZ`. With this timebase, you can generate pulses with a delay and length between 20 μs and 160 s. The timing resolution will be lower than if you are using `ND_INTERNAL_20_MHZ` timebase.
- `ND_GATE` to any legal value listed in the `GPCTR_Change_Parameter` function description.
- `ND_GATE_POLARITY` to `ND_NEGATIVE`. A high-to-low transition of the gate signal initiates the pulse generation timing.

GPCTR_Set_Application

Continued

You can use the `GPCTR_Change_Parameter` function after calling `GPCTR_Set_Application` and before calling `GPCTR_Control` with **action** = `ND_PROGRAM` or `ND_PREPARE`.

If you want to provide your timebase, you can connect your timebase source to one of the PFI pins on the I/O connector and change `ND_SOURCE` and `ND_SOURCE_POLARITY` to the appropriate values.

You can also configure the other general-purpose counter for `ND_SINGLE_TRIG_PULSE_GRN` and set `ND_SOURCE` of this counter to `ND_OTHER_GPCTR_TC` if you want to generate pulses with delays and intervals longer than 160 s.

application = `ND_RETRIG_PULSE_GNR`

In this application, the counter is used for the generation of a retriggerable delayed pulse after each transition on the gate input. By default, you get this by using the 20 MHz internal timebase (`ND_INTERNAL_20_MHZ`), so the resolution of timing is 50 ns. By default, the counter counts down from `ND_COUNT_1` = 5 million to 0 for the delay time and then down from `ND_COUNT_2` = 10 million to 0 for the pulse generation time to generate a 0.5 s pulse after 0.25 s of delay. By default, the gate is the PFI9/GPCTR0_GATE I/O connector pin for general-purpose counter 0 and the PFI4/GPCTR1_GATE I/O connector pin for general-purpose counter 1, and the transition which initiates the pulse generation is low-to-high. All transitions of the gate signal after you arm the counter initiate pulse generation.

The default 20 MHz timebase, combined with the counter width (24 bits), lets you generate pulses with a delay and length between 100 ns and 0.8 s.

For example, assume that you want to generate a pulse 200 ns long after 150 ns of delay from every transition of the gate signal. You need to set `ND_COUNT_1` to 150 ns/50 ns = 3 and `ND_COUNT_2` to 200 ns/50 ns = 4. Figure 2-20 shows a counter used for `ND_RETRIG_PULSE_GNR` after the following programming sequence:

```
GPCTR_Control(deviceNumber, gpctrNum, ND_RESET)
GPCTR_Set_Application(deviceNumber, gpctrNum, ND_RETRIG_PULSE_GNR)
GPCTR_Change_Parameter(deviceNumber, gpctrNum, ND_COUNT_1, 3)
GPCTR_Change_Parameter(deviceNumber, gpctrNum, ND_COUNT_2, 4)
Select_Signal(deviceNumber, gpctrNumOut, gpctrNumOut,ND_LOW_TO_HIGH)
GPCTR_Control(deviceNumber, gpctrNum, ND_PROGRAM)
```

GPCTR_Set_Application

Continued

In Figure 2-20:

- Gate is the signal present at the counter gate input.
- Source is the signal present at the counter source input.
- Output is the signal present at the counter output.

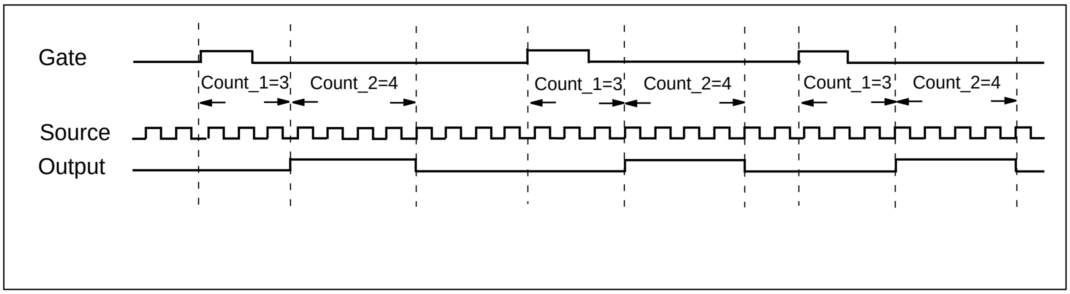


Figure 2-20. Retriggerable Pulse Generation

Use the `GPCTR_Control` function with **action** = `ND_RESET` to stop the pulse generation.

Typically, you will find modifying the following parameters through the `GPCTR_Change_Parameter` function useful when the counter **application** is `ND_RETRIG_PULSE_GNR`. You can change the following:

- `ND_COUNT_1` and `ND_COUNT_2` to any value between 2 and 2^{24} - 1. The defaults are given for illustrative purposes only.
- `ND_SOURCE` to `ND_INTERNAL_100_KHZ`. With this timebase, you can generate pulses with delay and length between 20 μs and 160 s. The timing resolution will be lower than if you are using `ND_INTERNAL_20_MHZ` timebase.
- `ND_GATE` to any legal value listed in the `GPCTR_Change_Parameter` function description.
- `ND_GATE_POLARITY` to `ND_NEGATIVE`. A high-to-low transition of the gate signal initiates the pulse generation timing.

You can use the `GPCTR_Change_Parameter` function after calling `GPCTR_Set_Application` and before calling `GPCTR_Control` with **action** = `ND_PROGRAM` or `ND_PREPARE`.

GPCTR_Set_Application

Continued

If you want to provide your timebase, you can connect your timebase source to one of the PFI pins on the I/O connector and change `ND_SOURCE` and `ND_SOURCE_POLARITY` to the appropriate values.

You can also configure the other general-purpose counter for `ND_RETRIG_PULSE_GNR`, and set `ND_SOURCE` of this counter to `ND_OTHER_GPCTR_TC` if you want to generate pulses with delays and intervals longer than 160 s.

application = `ND_PULSE_TRAIN_GNR`

In this application, the counter is used for generation of a pulse train. By default, you get this by using the 20 MHz internal timebase (`ND_INTERNAL_20_MHZ`), so the resolution of timing is 50 ns. By default, the counter repeatedly counts down from `ND_COUNT_1` = 5 million to 0 for the delay time and then down from `ND_COUNT_2` = 10 million to 0 for the pulse generation time to generate a train 0.5 s pulses separated by 0.25 s of delay. Pulse train generation starts as soon as you arm the counter. You must reset the counter to stop the pulse train.

The default 20 MHz timebase, combined with the counter width (24 bits), lets you generate trains consisting of pulses with delay and length between 100 ns and 0.8 s.

Assume that you want to generate a pulse train with the low period 150 ns long and the high period 200 ns long. You need to set `ND_COUNT_1` to 150 ns/50 ns = 3 and `ND_COUNT_2` to 200 ns/50 ns = 4. This corresponds to a 20 MHz : (3 + 4) = 2.86 MHz signal with (3/7)/(4/7) = 43/57 duty cycle. Figure 2-21 shows the scenario of a counter used for `ND_PULSE_TRAIN_GNR` after the following programming sequence:

```
GPCTR_Control(deviceNumber, gpctrNum, ND_RESET)

GPCTR_Set_Application(deviceNumber, gpctrNum, ND_PULSE_TRAIN_GNR)

GPCTR_Change_Parameter(deviceNumber, gpctrNum, ND_COUNT_1, 3)

GPCTR_Change_Parameter(deviceNumber, gpctrNum, ND_COUNT_2, 4)

Select_Signal(deviceNumber, gpctrNumOut, gpctrNumOut,ND_LOW_TO_HIGH)

GPCTR_Control(deviceNumber, gpctrNum, ND_PROGRAM)
```

In Figure 2-21

- Source is the signal present at the counter source input.
- Output is the signal present at the counter output.

GPCTR_Set_Application

Continued

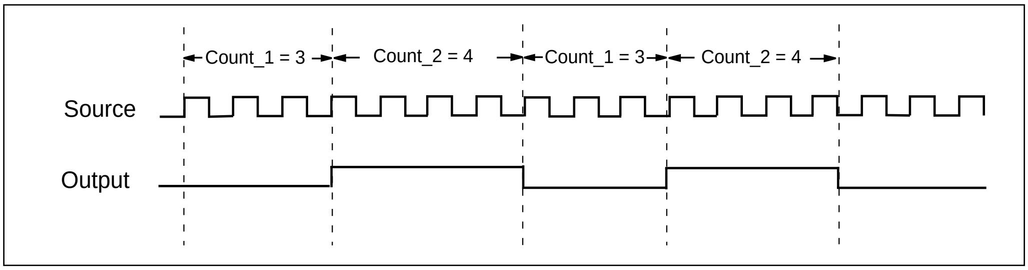


Figure 2-21. Pulse Train Generation

Use the `GPCTR_Control` function with **action** = `ND_RESET` to stop the pulse generation.

Typically, you will find modifying the following parameters through the `GPCTR_Change_Parameter` function useful when the counter **application** is `ND_PULSE_TRAIN_GNR`. You can change the following:

- `ND_COUNT_1` and `ND_COUNT_2` to any value between 2 and 2^{24} - 1. The defaults are given for illustrative purposes only.
- `ND_SOURCE` to `ND_INTERNAL_100_KHZ`. With this timebase, you can generate pulses with delay and length between 20 μs and 160 s. The timing resolution will be lower than if you are using the `ND_INTERNAL_20_MHZ` timebase.

You can use the `GPCTR_Change_Parameter` function after calling `GPCTR_Set_Application` and before calling `GPCTR_Control` with **action** = `ND_PROGRAM` or `ND_PREPARE`.

If you want to provide your timebase, you can connect your timebase source to one of the PFI pins on the I/O connector and change `ND_SOURCE` and `ND_SOURCE_POLARITY` to the appropriate values.

You can also configure the other general-purpose counter for `ND_PULSE_TRAIN_GNR`, and set `ND_SOURCE` of this counter to `ND_OTHER_GPCTR_TC` if you want to generate pulses with delays and intervals longer than 160 s.

GPCTR_Set_Application

Continued

application = ND_FSK

In this application, the counter is used for generation of frequency shift keyed signals. The counter generates a pulse train of one frequency and duty cycle when the gate is low, and a pulse train with different parameters when the gate is high. By default, you get this by using the 20 MHz internal timebase (ND_INTERNAL_20_MHZ), so the resolution of timing is 50 ns. By default, when the gate is low, the counter repeatedly counts down from ND_COUNT_1 = 5 million to 0 for the delay time, and then down from ND_COUNT_2 = 10 million to 0 for the pulse generation time, to generate a train 0.5 s pulses separated by 0.25 s of delay. Also by default, when the gate is high, the counter repeatedly counts down from ND_COUNT_3 = 4 million to 0 for the delay time, and then down from ND_COUNT_4 = 6 million to 0 for the pulse generation time, to generate a train 0.3 s pulses separated by 0.2 s of delay. The FSK pulse generation starts as soon as you arm the counter. You must reset the counter to stop the pulse generation.

The default 20 MHz timebase, combined with the counter width (24 bits), lets you generate pulses with a delay and length between 100 ns and 0.8 s.

Assume that you want to generate a pulse train with 100 ns low and 150 ns high time when the gate is low and with 300 ns low time and 200 ns high time when the gate is high. You need to set ND_COUNT_1 to 100 ns/50 ns = 2, ND_COUNT_2 to 150 ns/50 ns = 3, ND_COUNT_3 to 300 ns/50 ns = 6, and ND_COUNT_4 to 200 ns/50 ns = 4. Figure 2-22 shows a counter used for ND_FSK after the following programming sequence:

```
GPCTR_Control(deviceNumber, gpctrNum, ND_RESET)
GPCTR_Set_Application(deviceNumber, gpctrNum, ND_RETRIG_PULSE_GNR)
GPCTR_Change_Parameter(deviceNumber, gpctrNum, ND_COUNT_1, 2)
GPCTR_Change_Parameter(deviceNumber, gpctrNum, ND_COUNT_2, 3)
GPCTR_Change_Parameter(deviceNumber, gpctrNum, ND_COUNT_3, 6)
GPCTR_Change_Parameter(deviceNumber, gpctrNum, ND_COUNT_4, 4)
Select_Signal(deviceNumber, gpctrNumOut, gpctrNumOut,ND_LOW_TO_HIGH)
GPCTR_Control(deviceNumber, gpctrNum, ND_PROGRAM)
```

In Figure 2-22:

- Gate is the signal present at the counter gate input.
- Source is the signal present at the counter source input.
- Output is the signal present at the counter output.

GPCTR_Set_Application

Continued

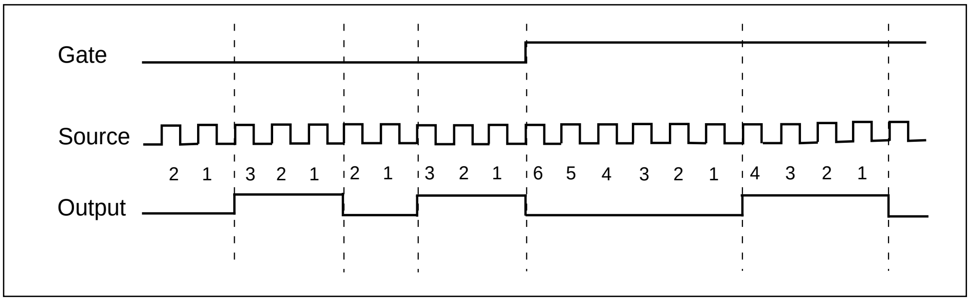


Figure 2-22. Frequency Shift Keying

Use the `GPCTR_Control` function with **action** = `ND_RESET` to stop the pulse generation.

Typically, you will find modifying the following parameters through the `GPCTR_Change_Parameter` function useful when the counter **application** is `ND_RETRIG_PULSE_GNR`. You can change the following:

- `ND_COUNT_1`, `ND_COUNT_2`, `ND_COUNT_3`, and `ND_COUNT_4` to any value between 2 and 2^{24} - 1. The defaults are given for illustrative purposes only.
- `ND_SOURCE` to `ND_INTERNAL_100_KHZ`. With this timebase, you can generate pulses with a delay and length between 20 μs and 160 s. The timing resolution will be lower than if you are using the `ND_INTERNAL_20_MHZ` timebase.
- `ND_GATE` to any legal value listed in the `GPCTR_Change_Parameter` function description.

You can use the `GPCTR_Change_Parameter` function after calling `GPCTR_Set_Application` and before calling `GPCTR_Control` with **action** = `ND_PROGRAM` or `ND_PREPARE`.

If you want to provide your timebase, connect your timebase source to one of the PFI pins on the I/O connector and change `ND_SOURCE` and `ND_SOURCE_POLARITY` to the appropriate values.

You can also configure the other general-purpose counter for `ND_FSK`, and set `ND_SOURCE` of this counter to `ND_OTHER_GPCTR_TC` if you want to generate pulses with delays and intervals longer than 160 s.

GPCTR_Set_Application

Continued

application = `ND_BUFFERED_EVENT_CNT`

In this application, the counter is used for continuous counting of events. By default, the events are low-to-high transitions on the PFI8/GPCTR0_SOURCE I/O connector pin for general-purpose counter 0 and the PFI3/GPCTR1_SOURCE I/O connector pin for general-purpose counter 1. Counts present at specified events of the signal present at the gate are saved in a buffer. By default, those events are the low-to-high transitions of the signal on the PFI9/GPCTR0_GATE I/O connector pin for general-purpose counter 0 and the PFI4/GPCTR1_GATE I/O connector pin for general-purpose counter 1. The counter counts up starting from 0; its contents are placed in the buffer after an edge of appropriate polarity is detected on the gate; the counter keeps counting without interruption. NI-DAQ transfers data from the counter into the buffer until the buffer is filled; the counter is disarmed at that time.

The counter width (24 bits) lets you count up to 2^{24}-1 events. Figure 2-23 shows one possible scenario of a counter used for `ND_BUFFERED_EVENT_CNT` after the following programming sequence:

```
Make buffer be a 100-element array of U32.
GPCTR_Control(deviceNumber, gpctrNum, ND_RESET)
GPCTR_Set_Application(deviceNumber, gpctrNum, ND_BUFFERED_EVENT_CNT)
GPCTR_Config_Buffer(deviceNumber, gpctrNum, 0, 100, buffer)
GPCTR_Control(deviceNumber, gpctrNum, ND_PROGRAM)
```

In Figure 2-23:

- Gate is the signal present at the counter gate input.
- Source is the signal present at the counter source input.
- Buffer is the contents of the buffer; you can retrieve data from the buffer when the counter becomes disarmed.

GPCTR_Set_Application

Continued

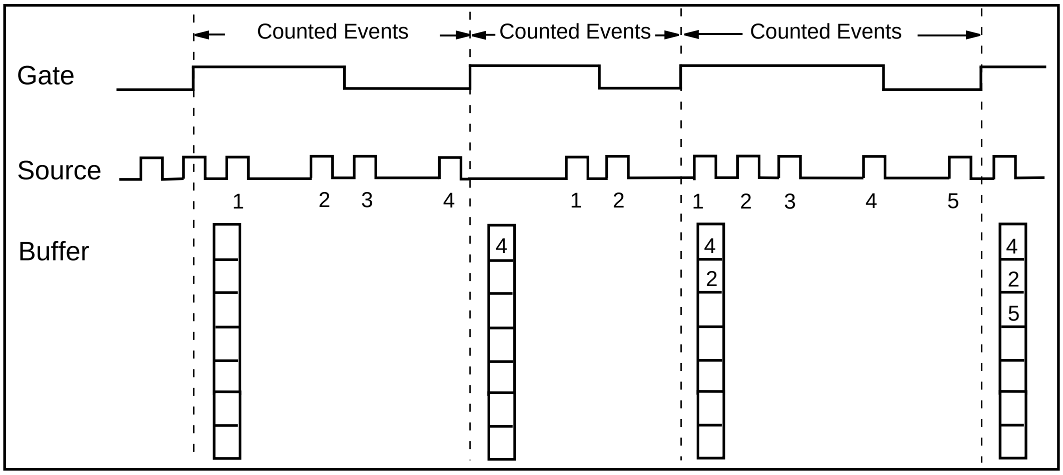


Figure 2-23. Buffered Event Counting

Use the `GPCTR_Watch` function with **entityID** = `ND_ARMED` to monitor the progress of the counting process. This measurement completes when **entityValue** becomes `ND_NO`. You can do this as follows:

```
Create U32 variable counter_armed.
repeat
{
       GPCTR_Watch(deviceNumber, gpctrNumber, ND_ARMED, counter_armed)
}
until (counter_armed = ND_NO)
```

When the counter is disarmed, you can safely access data in the buffer.

Typically, you will find modifying the following parameters through the `GPCTR_Change_Parameter` function useful when the counter **application** is `ND_BUFFERED_PERIOD_MSR`. You can change the following:

- `ND_SOURCE` to any legal value listed in the `GPCTR_Change_Parameter` function description.
- `ND_SOURCE_POLARITY` to `ND_HIGH_TO_LOW`.

GPCTR_Set_Application

Continued

- `ND_GATE` to any legal value listed in the `GPCTR_Change_Parameter` function description.
- `ND_GATE_POLARITY` to `ND_NEGATIVE`. Counts will be captured on every high-to-low transition of the signal present at the gate.

☞ **Note:** ***The counter will start counting as soon as you arm it. However, it will not count if the gate signal stays in low state when*** `ND_GATE_POLARITY` ***is*** `ND_POSITIVE` ***or if it stays in high state when*** `ND_GATE_POLARITY` ***is*** `ND_NEGATIVE` ***while*** `GPCTR_Control` ***is executed with*** **action** = `ND_ARM` ***or*** **action** = `ND_PROGRAM`***. Be aware of this when you interpret the first count in your buffer.***

application = `ND_BUFFERED_PERIOD_MSR`

In this application, the counter is used for continuous measurement of the time interval between successive transitions of the same polarity of the gate signal. By default, those are the low-to-high transitions of the signal on the PFI9/`GPCTR0_GATE` I/O connector pin for general-purpose counter 0 and the PFI4/`GPCTR1_GATE` I/O connector pin for general-purpose counter 1. The counter counts the 20 MHz internal timebase (`ND_INTERNAL_20_MHZ`), so the resolution of measurement is 50 ns. The counter counts up starting from 0; its contents are placed in the buffer after an edge of appropriate polarity is detected on the gate; the counter then starts counting up from 0 again. NI-DAQ transfers data from the counter into the buffer until the buffer is filled; the counter is disarmed at that time.

The default 20 MHz timebase, combined with the counter width (24 bits), lets you measure the width of a pulse between 100 ns and 0.8 s long.

Figure 2-24 shows one possible scenario of a counter used for `ND_BUFFERED_PERIOD_MSR` after the following programming sequence:

```
Make buffer be a 100-element array of U32.

GPCTR_Control(deviceNumber, gpctrNum, ND_RESET)

GPCTR_Set_Application(deviceNumber, gpctrNum, ND_BUFFERED_PERIOD_MSR)

GPCTR_Config_Buffer(deviceNumber, gpctrNum, 0, 100, buffer)

GPCTR_Control(deviceNumber, gpctrNum, ND_PROGRAM)
```

GPCTR_Set_Application

Continued

In Figure 2-24:

- Gate is the signal present at the counter gate input.
- Source is the signal present at the counter source input.
- Buffer is the contents of the buffer; you can retrieve data from the buffer when the counter becomes disarmed.

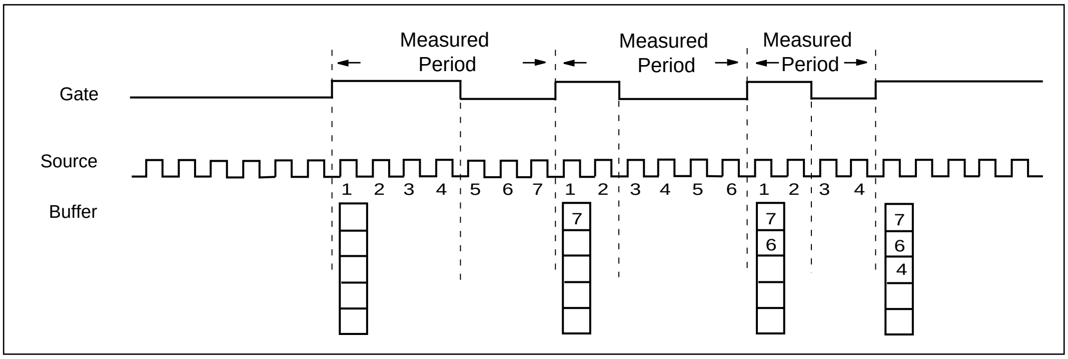


Figure 2-24. Buffered Period Measurement

Use the `GPCTR_Watch` function with **entityID** = `ND_ARMED` to monitor the progress of the counting process. This measurement completes when **entityValue** becomes `ND_NO`, as shown in the following example:

```
Create U32 variable counter_armed.

repeat

{

      GPCTR_Watch(deviceNumber, gpctrNumber, ND_ARMED, counter_armed)

}

until (counter_armed = ND_NO)
```

When the counter is disarmed, you can safely access data in the buffer.

GPCTR_Set_Application

Continued

Typically, you will find modifying the following parameters through the GPCTR_Change_Parameter function useful when the counter **application** is ND_BUFFERED_PERIOD_MSR. You can change the following:

- ND_SOURCE to ND_INTERNAL_100_KHZ. With this timebase, you can measure intervals between 20 µs and 160 s long. The resolution will be lower than if you are using ND_INTERNAL_20_MHZ timebase.
- ND_SOURCE_POLARITY to ND_HIGH_TO_LOW.
- ND_GATE to any legal value listed in the GPCTR_Change_Parameter function description.
- ND_GATE_POLARITY to ND_NEGATIVE. Measurements will be performed between successive high-to-low transitions of the signal present at the gate.

If you want to provide your timebase, you can connect your timebase source to one of the PFI pins on the I/O connector and change ND_SOURCE and ND_SOURCE_POLARITY to the appropriate values.

You can also configure the other general-purpose counter for ND_PULSE_TRAIN_GNR, and set ND_SOURCE of this counter to ND_OTHER_GPCTR_TC if you want to measure intervals longer than 160 s.

☞ **Note:** ***The counter will start counting as soon as you arm it. Be aware of this when you interpret the first count in your buffer.***

application = ND_BUFFERED_SEMI_PERIOD_MSR

In this application, the counter is used for the continuous measurement of the time interval between successive transitions of the gate signal. By default, those are all transitions of the signal on the PFI9/GPCTR0_GATE I/O connector pin for general-purpose counter 0 and the PFI4/GPCTR1_GATE I/O connector pin for general-purpose counter 1. The counter counts the 20 MHz internal timebase (ND_INTERNAL_20_MHZ), so the resolution of measurement is 50 ns. The counter counts up starting from 0; its contents are placed in the buffer after an edge is detected on the gate; the counter then starts counting up from 0 again. NI-DAQ transfers data from the counter into the buffer until the buffer is filled; the counter is disarmed at that time.

The default 20 MHz timebase, combined with the counter width (24 bits), lets you measure the width of a pulse between 100 ns and 0.8 s long.

GPCTR_Set_Application

Continued

Figure 2-25 shows one possible scenario of a counter used for `ND_BUFFERED_SEMI_PERIOD_MSR` after the following programming sequence:

```
Make buffer be a 100-element array of U32.

GPCTR_Control(deviceNumber, gpctrNum, ND_RESET)

GPCTR_Set_Application(deviceNumber, gpctrNum,
ND_BUFFERED_SEMI_PERIOD_MSR)

GPCTR_Config_Buffer(deviceNumber, gpctrNum, 0, 100, buffer)

GPCTR_Control(deviceNumber, gpctrNum, ND_PROGRAM)
```

In Figure 2-25:

- Gate is the signal present at the counter gate input.
- Source is the signal present at the counter source input.
- Buffer is the contents of the buffer; you can retrieve data from the buffer when the counter becomes disarmed.

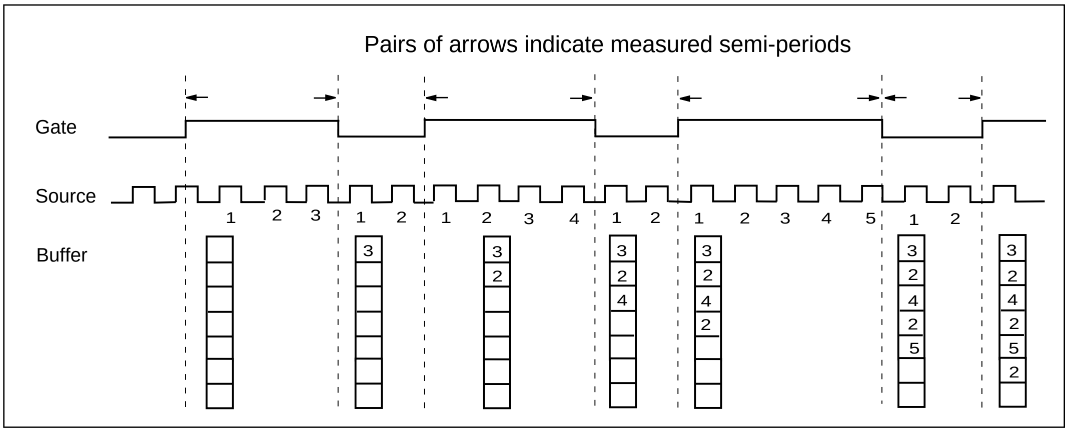


Figure 2-25. Buffered Semi-Period Measurement

Use the `GPCTR_Watch` function with **entityID** = `ND_ARMED` to monitor the progress of the counting process. This measurement completes when **entityValue** becomes `ND_NO`. The following code example shows this process:

```
Create U32 variable counter_armed.

repeat
```

GPCTR_Set_Application

Continued

```
{
      GPCTR_Watch(deviceNumber, gpctrNumber, ND_ARMED, counter_armed)
}
until (counter_armed = ND_NO)
```

When the counter is disarmed, you can safely access data in the buffer.

Typically, you will find modifying the following parameters through the `GPCTR_Change_Parameter` function useful when the counter **application** is `ND_BUFFERED_SEMI_PERIOD_MSR`. You can change the following:

- `ND_SOURCE` to `ND_INTERNAL_100_KHZ`. With this timebase, you can measure intervals between 20 μs and 160 s long. The resolution will be lower than if you are using the `ND_INTERNAL_20_MHZ` timebase.
- `ND_SOURCE_POLARITY` to `ND_HIGH_TO_LOW`.
- `ND_GATE` to any legal value listed in the `GPCTR_Change_Parameter` function description.

If you want to provide your timebase, you can connect your timebase source to one of the PFI pins on the I/O connector and change `ND_SOURCE` and `ND_SOURCE_POLARITY` to the appropriate values.

You can also configure the other general-purpose counter for `ND_PULSE_TRAIN_GNR` and set `ND_SOURCE` of this counter to `ND_OTHER_GPCTR_TC` if you want to measure intervals longer than 160 s.

☞ **Note:** ***The counter will start counting as soon as you arm it. Be aware of this when you interpret the first count in your buffer.***

application = `ND_BUFFERED_PULSE_WIDTH_MSR`
In this application, the counter is used for continuous measurement of width of pulses of selected polarity present at the counter gate. By default, those pulses are active high pulses present on the signal on the PFI9/GPCTR0_GATE I/O connector pin for general-purpose counter 0 and the PFI4/GPCTR1_GATE I/O connector pin for general-purpose counter 1. The counter counts the 20 MHz internal timebase (`ND_INTERNAL_20_MHZ`), so the resolution of measurement is 50 ns. The counter counts up starting from 0; its contents are placed in the buffer after a pulse completes; the counter then starts counting up from 0 again when the next pulse appears. NI-DAQ

GPCTR_Set_Application

Continued

transfers data from the counter into the buffer until the buffer is filled; the counter is disarmed at that time.

The default 20 MHz timebase, combined with the counter width (24 bits), lets you measure the width of a pulse between 100 ns and 0.8 s long.

When using the buffered counter operations in applications involving period/pulse-width measurements, the first acquired point may represent bad data. The first data point is the measured interval between the instant when the counter is armed and when the first edge transition takes place on the counter GATE. Because there is no deterministic way of specifying when the counter is actually armed, the first value may be incorrect. Subsequent data points acquired will not have this problem.

Figure 2-26 shows one possible scenario of a counter used for `ND_BUFFERED_PULSE_WIDTH_MSR` after the following programming sequence:

```
Make buffer be a 100-element array of U32.

GPCTR_Control(deviceNumber, gpctrNum, ND_RESET)

GPCTR_Set_Application(deviceNumber, gpctrNum,
ND_BUFFERED_PULSE_WIDTH_MSR)

GPCTR_Config_Buffer(deviceNumber, gpctrNum, 0, 100, buffer)

GPCTR_Control(deviceNumber, gpctrNum, ND_PROGRAM)
```

In Figure 2-26:

- Gate is the signal present at the counter gate input.
- Source is the signal present at the counter source input.
- Buffer is the contents of the buffer; you can retrieve data from the buffer when the counter becomes disarmed.

GPCTR_Set_Application

Continued

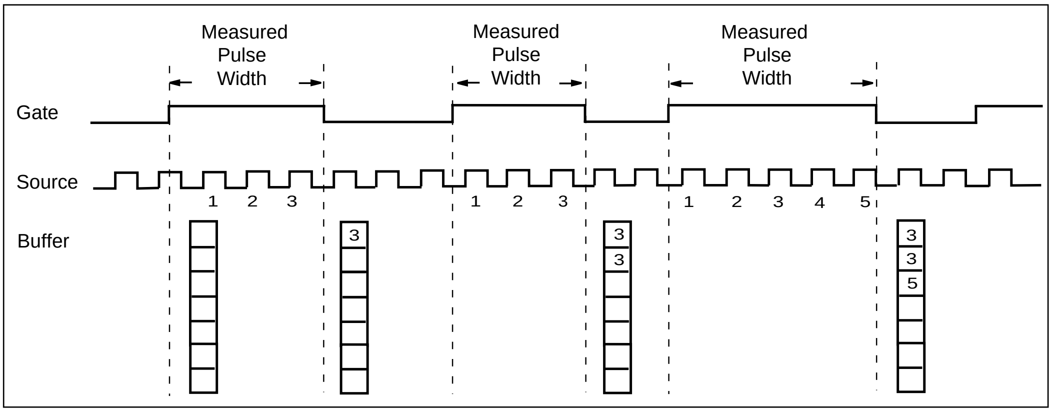


Figure 2-26. Buffered Pulse Width Measurement

Use the `GPCTR_Watch` function with **entityID** = `ND_ARMED` to monitor the progress of the counting process.

This measurement completes when **entityValue** becomes `ND_NO`. You can do this as follows:

```
Create U32 variable counter_armed.

repeat

{

      GPCTR_Watch(deviceNumber, gpctrNumber, ND_ARMED, counter_armed)

}

until (counter_armed = ND_NO)
```

When the counter is disarmed, you can safely access data in the buffer.

Typically, you will find modifying the following parameters through the `GPCTR_Change_Parameter` function useful when the counter **application** is `ND_BUFFERED_PULSE_WIDTH_MSR`. You can change the following:

- `ND_SOURCE` to `ND_INTERNAL_100_KHZ`. With this timebase, you can measure intervals between 20 μs and 160 s long. The resolution will be lower than if you are using `ND_INTERNAL_20_MHZ` timebase.

GPCTR_Set_Application

Continued

- `ND_SOURCE_POLARITY` to `ND_HIGH_TO_LOW`.
- `ND_GATE` to any legal value listed in the `GPCTR_Change_Parameter` function description.
- `ND_GATE_POLARITY` to `ND_NEGATIVE`. Measurements will be performed on the active low pulses.

If you want to provide your timebase, you can connect your timebase source to one of the PFI pins on the I/O connector and change `ND_SOURCE` and `ND_SOURCE_POLARITY` to the appropriate values.

You can also configure the other general-purpose counter for `ND_PULSE_TRAIN_GNR`, and set `ND_SOURCE` of this counter to `ND_OTHER_GPCTR_TC` if you want to measure intervals longer than 160 s.

Note: ***You must make sure that there is at least one source transition during the measured pulse in order for this application to work properly.***

GPCTR_Watch

Format

status = GPCTR_Watch (deviceNumber, gpctrNum, entityID, entityValue)

Purpose

Monitors state of the general-purpose counter and its operation (E Series devices only).

☞ **Note:** ***NI-DAQ for Windows NT does not support this function.***

Parameters

Input

Name	Type		Description
	Windows	Windows NT	
deviceNumber	I16	n/a	assigned by configuration utility
gpctrNum	U32	n/a	number of the counter you want to use
entityID	U32	n/a	identification of the feature you want to monitor

Output

Name	Type		Description
	Windows	Windows NT	
entityValue	U32	n/a	the value of the feature specified by **entityID**

GPCTR_Watch

Continued

Parameter Discussion

Legal ranges for the **gpctrNum**, **entityID**, and **entityValue** are in terms of constants defined in a header file. The header file you should use depends on the language you are using:

- C programmers—`NIDAQCNS.H` (`DATAACQ.H` for LabWindows/CVI)
- BASIC programmers—`NIDAQCNS.INC` (Visual Basic for Windows programmers should refer to the Programming Language Considerations section in Chapter 1 for more information.)
- Pascal programmers—`NIDAQCNS.PAS`

Use **gpctrNum** to indicate to NI-DAQ which counter you want to program. Legal values for this parameter are `ND_COUNTER_0` and `ND_COUNTER_1`.

Use **entityID** to indicate to NI-DAQ which feature you are interested in. Legal values are listed in the following paragraphs, along with the corresponding values you can expect for **entityValue**. **entityValue** will be given either in terms of constants from the header file, or as numbers, as appropriate.

entityID = `ND_COUNT`
This is the counter contents. **entityValue** can be between 0 and 2^{24}-1.

entityID = `ND_ARMED`
Indicates whether the counter is armed. **entityValue** can be `ND_YES` or `ND_NO`. You can use this in applications such as `ND_SINGLE_PULSE_WIDTH_MSR` for finding out when the pulse width measurement completes.

entityID = `ND_TC_REACHED`
Indicates whether the counter has reached terminal count **entityValue** can be `ND_YES` or `ND_NO`. You can use this in applications such as `ND_SINGLE_PULSE_WIDTH_MSR` for detecting overflow (pulse was too long to be measured using the selected timebase).

entityID = `ND_DONE`
When the application is `ND_SINGLE_TRIG_PULSE_GNR`, this indicates that the pulse has completed. When the application is `ND_RETRIG_PULSE_GNR`, this indicates that an individual pulse has completed. In this case, the indication that an individual pulse has completed will be returned only once per pulse by the `GPCTR_Watch` function.

entityID = `ND_OUTPUT_STATE`
You can use this to read the value of the counter output; the range is `ND_LOW` and `ND_HIGH`.

Note: ***C Programmers*—entityValue *is a pass-by-reference parameter.***

ICTR_Read

Format

status = ICTR_Read (deviceNumber, ctr, count)

Purpose

Reads the current contents of the selected counter without disturbing the counting process and returns the count.

Parameters

Input

Name	Type		Description
	Windows	Windows NT	
deviceNumber	I16	I32	assigned by configuration utility
ctr	I16	I32	counter number

Output

Name	Type		Description
	Windows	Windows NT	
count	U16	U32	current count

Parameter Discussion

ctr is the counter number.
Range: 0 through 2.

count returns the current count of the specified counter while the counter is counting down. **count** can be between zero and 65,535 when **ctr** is configured in binary mode (the default). **count** can be between zero and 9,999 if the last call to `ICTR_Setup` configured **ctr** in BCD counting mode.

ICTR_Read

Continued

☞ **Note:** ***C Programmers*—count *is a pass-by-reference parameter.***

☞ **Note:** ***BASIC Programmers—NI-DAQ returns count as a 16-bit unsigned number. In BASIC, integer variables are represented by a 16-bit two's complement system. Thus, values greater than 32,767 are incorrectly treated as negative numbers. You can avoid this problem by using a long number as shown below:***

```
if count% < 0 then
      lcount& = count% + 65,536
else
      lcount& = count%
end if
```

ICTR_Reset

Format

status = ICTR_Reset (deviceNumber, ctr, state)

Purpose

Sets the output of the selected counter to the specified state.

Parameters

Input

Name	Type		Description
	Windows	Windows NT	
deviceNumber	I16	I32	assigned by configuration utility
ctr	I16	I32	counter number
state	I16	I32	logic state to be reset

Parameter Discussion

ctr is the counter number.
Range: 0 through 2.

state is the logic state to which the counter is to be reset.
Range: 0 or 1.

If **state** is 0, the counter output is forced low by programming the specified counter in Mode 0. NI-DAQ does *not* load the count register; thus, the output remains low until NI-DAQ programs the counter in another mode. If **state** is 1, NI-DAQ forces the counter output high by programming the given counter in Mode 2. NI-DAQ does not load the count register; thus, the output remains high until NI-DAQ programs the counter in another mode.

ICTR_Setup

Format

status = ICTR_Setup (deviceNumber, ctr, mode, count, binBcd)

Purpose

Configures the given counter to operate in the specified mode.

Parameters

Input

Name	Type		Description
	Windows	Windows NT	
deviceNumber	I16	I32	assigned by configuration utility
ctr	I16	I32	counter number
mode	I16	I32	mode in which the counter is to operate
count	U16	U32	period from one output pulse to the next
binBcd	I16	I32	16-bit binary or 4-decade binary-coded decimal

Parameter Discussion

ctr is the counter number.
Range: 0 through 2.

mode is the mode in which the counter is to operate.

0: Toggle output from low to high on terminal count.
1: Programmable one-shot.
2: Rate generator.

ICTR_Setup

Continued

3: Square wave rate generator.
4: Software-triggered strobe.
5: Hardware-triggered strobe.

In Mode 0, the output goes low after the mode set operation, and the counter begins to count down while the gate input is high. The output goes high when NI-DAQ reaches the terminal count (that is, the counter has decremented to zero) and stays high until you set the selected counter to a different mode. Figure 2-27 shows the Mode 0 timing diagram.

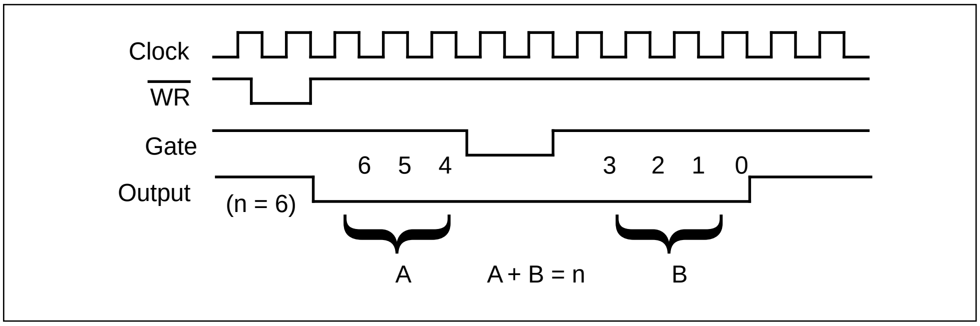


Figure 2-27. Mode 0 Timing Diagram

In Mode 1, the output goes low on the count following the rising edge of the gate input and goes high on terminal count. Figure 2-28 shows the Mode 1 timing diagram.

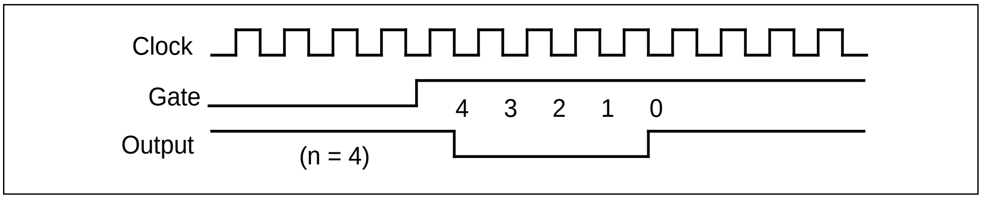


Figure 2-28. Mode 1 Timing Diagram

ICTR_Setup

Continued

In Mode 2, the output goes low for one period of the clock input. **count** indicates the period from one output pulse to the next. Figure 2-29 shows the Mode 2 timing diagram.

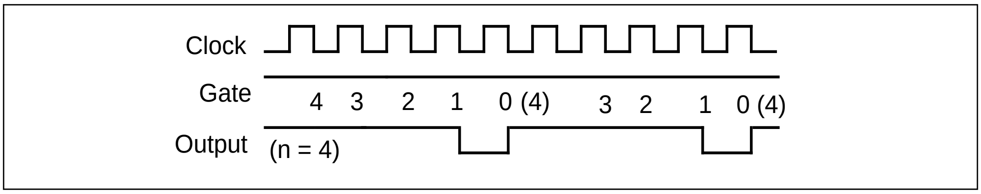


Figure 2-29. Mode 2 Timing Diagram

In Mode 3, the output stays high for one half of the **count** clock pulses and stays low for the other half. Figure 2-30 shows the Mode 3 timing diagram.

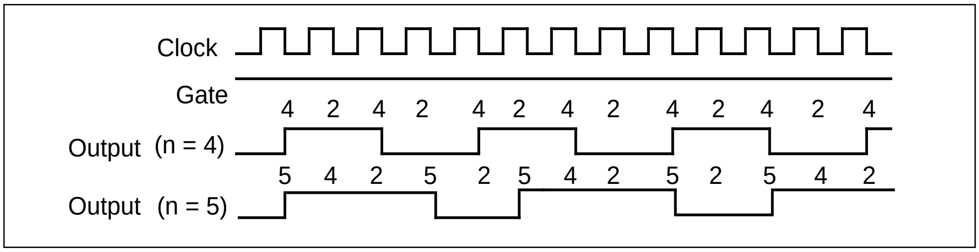


Figure 2-30. Mode 3 Timing Diagram

In Mode 4, the output is initially high, and the counter begins to count down while the gate input is high. On terminal count, the output goes low for one clock pulse, then goes high again. Figure 2-31 shows the Mode 4 timing diagram.

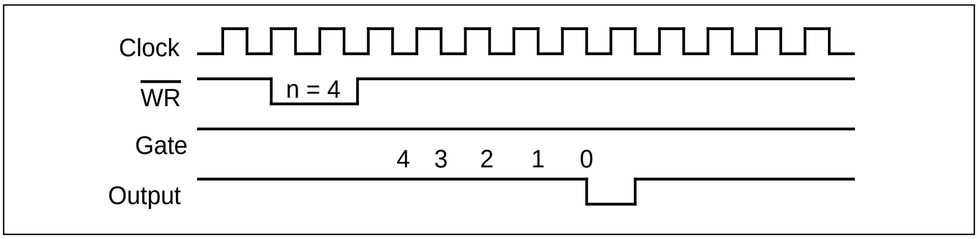


Figure 2-31. Mode 4 Timing Diagram

ICTR_Setup

Continued

Mode 5 is similar to Mode 4 except that the gate input is used as a trigger to start counting. Figure 2-32 shows Mode 5 timing diagram.

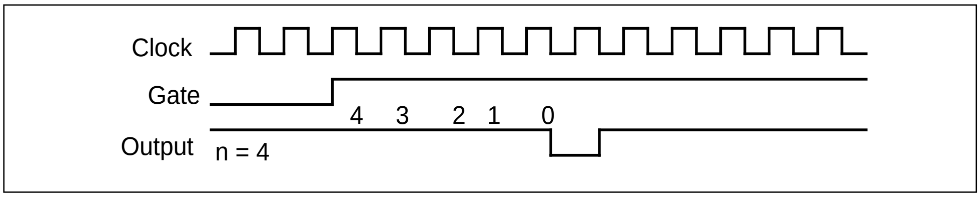


Figure 2-32. Mode 5 Timing Diagram

See the 8253 Programmable Interval Timer data sheet in your DAQCard-500/700 or Lab and 1200 series user manual for a detailed description of these modes and the associated timing diagrams.

count is the period from one output pulse to the next.
Range for Modes 0, 1, 4 and 5:
0 through 65,535 in binary counter operation.
0 through 9,999 in BCD counter operation.
Range for Modes 2 and 3:
2 through 65,535 and 0 in binary counter operation.
2 through 9,999 and 0 in BCD counter operation.

☞ **Note:** ***Zero is equivalent to 65,536 in binary counter operation and 10,000 in BCD counter operation.***

☞ **Note:** ***BASIC Programmers—NI-DAQ passes count as a 16-bit unsigned number. In BASIC, integer variables are represented by a 16-bit two's complement system. Thus,*** **count** ***values greater than 32,767 must be passed as negative numbers. One way to obtain the*** **count** ***value to be passed is to assign the required number between zero and 65,535 to a long variable and then obtain*** **count** ***as shown below:***

```
count% = lcount& - 65,536
```

binBcd controls whether the counter operates as a 16-bit binary counter or as a 4-decade binary-coded decimal (BCD) counter.

0:	4-decade BCD counter.
1:	16-bit binary counter.

Init_DA_Brds

Format

status = Init_DA_Brds (deviceNumber, deviceNumberCode)

Purpose

Initializes the hardware and software states of a National Instruments DAQ device to its default state, and then returns a numeric device code that corresponds to the type of device initialized. Any operation that the device is performing is halted. This function is called automatically and does not have to be explicitly called by your application. This function is useful for reinitializing the device hardware, for reinitializing the NI-DAQ software, and for determining which device has been assigned to a particular slot number. `Init_DA_Brds` will clear all configured messages for the device just as if you called `Config_DAQ_Event_Message` with a mode of 0.

Parameters

Input

Name	Type		Description
	Windows	Windows NT	
deviceNumber	I16	I32	assigned by configuration utility

Output

Name	Type		Description
	Windows	Windows NT	
deviceNumberCode	I16	I32	type of device

Init_DA_Brds

Continued

Parameter Discussion

deviceNumberCode indicates the type of device initialized.

-1:	Not a National Instruments DAQ device.
0:	AT-MIO-16L-9.
1:	AT-MIO-16L-15.
2:	AT-MIO-16L-25.
4:	AT-MIO-16H-9.
5:	AT-MIO-16H-15.
6:	AT-MIO-16H-25.
7:	PC-DIO-24.
8:	AT-DIO-32F.
10:	EISA-A2000.
11:	AT-MIO-16F-5.
12:	PC-DIO-96/PnP.
13:	PC-LPM-16.
14:	PC-TIO-10.
15:	AT-AO-6.
16:	AT-A2150S.
17:	AT-DSP2200 (64 Kword version).
18:	AT-DSP2200 (128/256/384 Kword version).
19:	AT-MIO-16X.
20:	AT-MIO-64F-5.
21:	AT-MIO-16DL-9.
22:	AT-MIO-16DL-25.
23:	AT-MIO-16DH-9.
24:	AT-MIO-16DH-25.
25:	AT-MIO-16E-2.
26:	AT-AO-10.
27:	AT-A2150C.
28:	Lab-PC+.
30:	SCXI-1200.
31:	DAQCard-700.
32:	NEC-MIO-16E-4.
33:	DAQPad-1200.
35:	DAQCard-DIO-24.
36:	AT-MIO-16E-10.
37:	AT-MIO-16DE-10.
38:	AT-MIO-64E-3.

Init_DA_Brds

Continued

39:	AT-MIO-16XE-50.
40:	NEC-AI-16E-4.
41:	NEC-MIO-16XE-50.
42:	NEC-AI-16XE-50.
43:	DAQPad-MIO-16XE-50.
44:	AT-MIO-16E-1.
45:	PC-OPDIO-16.
46:	PC-AO-2DC.
47:	DAQCard-AO-2DC.
48:	DAQCard-1200.
49:	DAQCard-500
50:	AT-MIO-16XE-10.
51:	AT-AI-16XE-10.
52:	DAQCard-AI-16XE-50.
53:	DAQCard-AI-16E-4.
54:	DAQCard-516.
55:	PC-516.
56:	PC-LPM-16PnP.

☞ **Note:** ***C Programmers*—deviceNumberCode *is a pass-by-reference parameter.***

☞ **Notes:** ***(AT-DSP2200 only)*** `Init_DA_Brds`***( ) may take a few seconds to execute on an AT-DSP2200, depending on your computer speed, because*** `Init_DA_Brds(`***) loads the DSP kernel onto the DSP board.***

(AT-MIO-16X only) Calibration of the AT-MIO-16X takes up to 2 s. Therefore, `Init_DA_Brds()`***, which calls*** `MIO_Calibrate()`***, can take up to 2 s to execute.***

Using This Function

`Init_DA_Brds` initializes the device in the specified slot to the default conditions. These conditions are summarized for each device as follows:

- MIO and AI devices
  - Analog Input defaults:
    Mode = Differential.
    Range = 20 V (10 V for AT-MIO-16F-5, AT-MIO-64F-5, and 12-bit E Series).

Init_DA_Brds

Continued

Polarity = Bipolar(-10 V to +10 V for MIO-16, AT-MIO-16D, AT-MIO-16X, and 16-bit E Series devices; and -5 to +5 V for all other devices).
External conversion = Disabled.
Start trigger = Disabled.
Stop trigger = Disabled.
Gain and offset calibration values are loaded (AT-MIO-16F-5 only).

- Analog Output defaults (MIO devices only):
  Range = 20 V.
  Reference = 10 V.
  Mode = Bipolar (-10 to +10 V).
  Level = 0 V.
- Digital Input and Output defaults:
  Direction = Input.
  For ports 2, 3, and 4 of the AT-MIO-16D and AT-MIO-16DE-10, see also the default conditions of ports 0, 1, and 2 of the DIO-24.
- Counter/Timer defaults for Am9513-based MIO devices:
  Gating mode = No gating.
  Output type = Terminal count toggled (that is, TC toggled).
  Output polarity = Positive.
  Edge mode = Count rising edges.
  Count mode = Count once.
  Output level = Off. After you call `Init_DA_Brds`, the output of each counter is in a high-impedance state. Counter 1 on the MIO-16 and AT-MIO-16D, and counters 1, 2, and 5 on the AT-MIO-16F-5, AT-MIO-64F-5, and AT-MIO-16X are pulled up to +5 V while in the high-impedance state.

• DIO-24/DIO-32F/DIO-96

- Digital Input and Output defaults:
  Direction = Input.
  Handshaking = Disabled.
  Group assignments = No ports assigned to any group.

• PC-TIO-10

- Digital Input and Output defaults:
  Direction = Input.

Init_DA_Brds

Continued

- Counter/Timer defaults:
  Gating mode = No gating.
  Output type = Terminal count toggled.
  Output polarity = Positive.
  Edge mode = Count rising edges.
  Count mode = Count once.
  Output level = Off.

• Lab and 1200 series devices

- Analog Input defaults:
  Input mode = Single-ended (eight single-ended input channels).
  Polarity = Bipolar (-5 to +5 V).
  External conversion = Disabled.
  Start trigger = Disabled.
  Stop trigger = Disabled.
- Analog Output defaults:
  Mode = Bipolar (-5 to +5 V).
  Level = 0 V.
- Digital Input and Output defaults:
  Direction = Input.
  Handshaking = Disabled.
  Group assignments = No ports assigned to any group.
- Counter/Timer defaults:
  Output level = Logical low.

• 516 and LPM devices and DAQCard-500/700

- Analog Input default:
  Mode = Single-ended (Differential also possible for 516 devices and DAQCard-700).
  Range = 10 V.
  Polarity = Bipolar (-5 to +5 V).
  External conversion = Disabled.
  Calibrated.
- Digital Input and Output defaults:
  Output port voltage level = 0 V.
- Counter/Timer defaults:
  Output level = Logical low.

Init_DA_Brds

Continued

- AT-AO-6/10
  - Analog Output defaults:
    Range = 20 V.
    Reference = 10 V.
    Mode = Bipolar (-10 to +10 V).
    Level = 0 V.
    Group assignments = No channels assigned to any group.
    Digital input and output defaults direction = Input.
- EISA-A2000
  - Analog Input defaults:
    Sample clock source = Internal.
    External conversion pin = Not driven.
    Dither = Off.
    Number of channels = 4.
    Sample interval = 20.
    Timebase = -1 (250 kHz sampling rate).
    Digital trigger = Disabled.
    Analog trigger = Disabled.
    Posttrigger delay = 0.
    Gain and offset calibration values are loaded.
  - Counter/Timer defaults:
    Gating mode = No gating.
    Output type = Terminal count toggled.
    Output polarity = Positive.
    Edge mode = Count rising edges.
    Count mode = Count once.
    Output level = Off.
- AT-A2150
  - Analog Input defaults:
    Sample clock source = Internal.
    External trigger pin = Not driven.
    Number of channels = 4.
    Digital trigger = Disabled.
    Analog trigger = Disabled.
    Posttrigger delay = 0.
  - Offset calibration performed using analog input ground.

Init_DA_Brds

Continued

- AT-DSP2200
  - Analog Input defaults:
    Sample clock source = Internal.
    External trigger pin = Not driven.
    Number of channels = 2.
    Digital trigger = Disabled.
    Analog trigger = Disabled.
    Posttrigger delay = 0.
    Translate and demux = Disabled.
  - Analog Output defaults:
    Level = 0 V.
    Translate = Disabled.
- AO-2DC devices
  - Analog Output defaults:
    Mode = Unipolar (0 to 10 V).
    Level = 0 V.
  - Digital Input and Output defaults:
    Direction = Input.

Of all these defaults, you can alter only the analog input and analog output settings of the MIO and AI devices, Lab-PC+, and PC-LPM-16 devices by setting jumpers on the device. If you have changed the jumpers from the factory settings, you must call either `AI_Configure` and/or `AO_Configure` after `Init_DA_Brds` so that the software copies of these settings reflect the true settings of the device.

If any device resources have been reserved for SCXI use when you make a call to `Init_DA_Brds`, those resources will still be reserved after you make the function call. Please refer to Chapter 15, *SCXI Hardware,* in the *DAQ Hardware Overview Guide* for listings of the different device resources that may be reserved for SCXI.

Lab_ISCAN_Check

Format

status = Lab_ISCAN_Check (deviceNumber, daqStopped, retrieved, finalScanOrder)

Purpose

Checks whether the current multiple-channel scanned data acquisition begun by the `Lab_ISCAN_Start` function is complete and returns the status, the number of samples acquired to that point, and the scanning order of the channels in the data array (DAQCard-500/700 and 516, Lab and 1200 series, and LPM devices only).

Parameters

Input

Name	**Type**		**Description**
	Windows	**Windows NT**	
deviceNumber	I16	I32	assigned by configuration utility

Output

Name	**Type**		**Description**
	Windows	**Windows NT**	
daqStopped	I16	I32	indicates whether the data acquisition has completed
retrieved	U32	U32	number of samples collected by the acquisition
finalScanOrder	[I16]	[I32]	the scan channel order

Lab_ISCAN_Check

Continued

Parameter Discussion

daqStopped returns an indication of whether the data acquisition has completed.

1: The data acquisition operation has stopped. Either NI-DAQ has acquired all the samples or an error has occurred.

0: The data acquisition operation is not yet complete.

retrieved indicates the progress of an acquisition. The meaning of **retrieved** depends on whether you have enabled pretrigger mode (see `DAQ_StopTrigger_Config`).

If pretrigger mode is disabled, **retrieved** returns the number of samples collected by the acquisition at the time of the call to `Lab_ISCAN_Check`. The value of **retrieved** increases until it equals the total number of samples to be acquired, at which time the acquisition terminates.

However, if pretrigger mode is enabled, **retrieved** returns the offset of the position in your buffer where NI-DAQ places the next data point when the function acquires. After the value of **retrieved** reaches **count** - 1 and rolls over to 0, the acquisition begins to overwrite old data with new data. When you apply a signal to the stop trigger input, the acquisition collects an additional number of samples specified by **ptsAfterStoptrig** in the call to `DAQ_StopTrigger_Config` and then terminates. When `Lab_ISCAN_Check` returns a status of 1, **retrieved** contains the offset of the oldest data point in the array (assuming that the acquisition has written to the entire buffer at least once). In pretrigger mode, `Lab_ISCAN_Check` automatically rearranges the array upon completion of the acquisition so that the oldest data point is at the beginning of the array. Thus, **retrieved** always equals 0 upon completion of a pretrigger mode acquisition. Since the stop trigger can occur in the middle of a scan sequence, the acquisition can end in the middle of a scan sequence. So, when the function rearranges the data in the buffer, the first sample may not belong to the first channel in the scan sequence. You can examine the **finalScanOrder** array to find out the way the data is arranged in the buffer.

finalScanOrder is an array that indicates the scan channel order of the data in the buffer passed to `Lab_ISCAN_Start`. The size of **finalScanOrder** must be at least equal to the number of channels scanned. This parameter is valid only when NI-DAQ returns **daqStopped** as 1 and is useful only when you enable pretrigger mode (Lab and 1200 series devices only).

If you do not use pretrigger mode, the values contained in **finalScanOrder** are, in single-ended mode, *n*-1, *n*-2, ...1, 0 to 0, in that order, and in differential mode, 2*(*n*-1),

Lab_ISCAN_Check

Continued

2*(*n*-2), ..., 2, 0, in that order, where *n* is the number of channels scanned. For example, if you scanned three channels in single-ended mode, the **finalScanOrder** returns:
finalScanOrder[0] = 2.
finalScanOrder[1] = 1.
finalScanOrder[2] = 0.

So the first sample in the buffer belongs to channel 2, the second sample belongs to channel 1, the third sample belong to channel 0, the fourth sample belongs to channel 2, and so on. This is the scan order expected from the Lab-PC+ and **finalScanOrder** is not useful in this case.

If you use pretrigger mode, the order of the channel numbers in **finalScanOrder** depends on where in the scan sequence the acquisition ended. This can vary because the stop trigger can occur in the middle of a scan sequence, which would cause the acquisition to end in the middle of a scan sequence so that the oldest data point in the buffer can belong to any channel in the scan sequence. `Lab_ISCAN_Check` rearranges the buffer so that the oldest data point is at index 0 in the buffer. This rearrangement causes the scanning order to change. This new scanning order is returned by **finalScanOrder**. For example, if you scanned three channels, the original scan order is channel 2, channel 1, channel 0, channel 2, channel 1, channel 0, and so on. However, after the stop trigger, if the acquisition ends after taking a sample from channel 1, the oldest data point belongs to channel 0. So **finalScanOrder** returns:
finalScanOrder[0] = 0.
finalScanOrder[1] = 2.
finalScanOrder[2] = 1.

So the first sample in the buffer belongs to channel 0, the second sample belongs to channel 2, the third sample belongs to channel 1, the fourth sample belongs to channel 0, and so on.

Note: ***C Programmers*—daqStopped *and* retrieved *are pass-by-reference parameters.***

Using This Function

`Lab_ISCAN_Check` checks the current background data acquisition operation to determine whether it has completed and returns the number of samples acquired at the time that you called `Lab_ISCAN_Check`. If the operation is complete, `Lab_ISCAN_Check` sets **daqStopped** = 1. Otherwise, **daqStopped** is set to 0. Before

Lab_ISCAN_Check

Continued

`Lab_ISCAN_Check` returns **daqStopped** = 1, it calls `DAQ_Clear`, allowing another Start call to execute immediately.

If `Lab_ISCAN_Check` returns an **overFlowError** or an **overRunError**, NI-DAQ has terminated the data acquisition operation because of lost A/D conversions due to a sample rate that is too high (sample interval was too small). An **overFlowError** indicates that the A/D FIFO memory overflowed because the data acquisition servicing operation was not able to keep up with sample rate. An **overRunError** indicates that the data acquisition circuitry was not able to keep up with the sample rate. Before returning either of these error codes, `Lab_ISCAN_Check` calls `DAQ_Clear` to terminate the operation and reinitialize the data acquisition circuitry.

Lab_ISCAN_Op

Format

status = Lab_ISCAN_Op (deviceNumber, numChans, gain, buffer, count, sampleRate, scanRate, finalScanOrder)

Purpose

Performs a synchronous, multiple-channel scanned data acquisition operation. `Lab_ISCAN_Op` does not return until NI-DAQ has acquired all the data or an acquisition error has occurred (DAQCard-500/700 and 516, Lab and 1200 series, and LPM devices only).

Parameters

Input

Name	Type		Description
	Windows	Windows NT	
deviceNumber	I16	I32	assigned by configuration utility
numChans	I16	I32	number of channels to be scanned
gain	I16	I32	gain setting
count	U32	U32	number of samples to be acquired
sampleRate	F64	F64	desired sample rate in units of pts/s
scanRate	F64	F64	desired scan rate in units of scans/s

Lab_ISCAN_Op

Continued

Output

Name	Type		Description
	Windows	Windows NT	
buffer	[I16], HDL	[I16], HDL	contains the acquired data
finalScanOrder	[I16]	[I32]	the scan channel order of the data

Parameter Discussion

numChans is the number of channels to be scanned in a single scans sequence. The value of this parameter also determines which channels NI-DAQ scans because these devices have a fixed scanning order. The scanned channels range from **numChans** - 1 to channel 0. If you are using SCXI modules with additional multiplexers, you must scan the analog input channels on the DAQ device that corresponds to the SCXI channels you want. You should select the SCXI scan list using `SCXI_SCAN_Setup` before you call this function. Please refer to the *NI-DAQ User Manual for PC Compatibles* for more information on SCXI channel assignments.

Range:	1 through 4 for the 516 and Lab and 1200 series devices in differential mode.
	1 through 8 for DAQCard-500 (single-ended mode only).
	1 through 8 for DAQCard-700 in differential mode.
	1 through 8 for the Lab and 1200 series devices in single-ended mode.
	1 through 16 for LPM devices or DAQCard-700 in single-ended mode.

gain is the gain setting to be used for the scanning operation. The same gain is applied to all the channels scanned. This gain setting applies only to the DAQ device; if you use SCXI modules with additional gain selection, you must establish any gain you want at the SCXI module either by setting jumpers on the module or by calling `SCXI_Set_Gain`. The following gain settings are valid for the Lab and 1200 series devices—1, 2, 5, 10, 20, 50, and 100. If you use an invalid gain, NI-DAQ returns an error. NI-DAQ ignores gain for the DAQCard-500/700 and 516 and LPM devices.

buffer is an integer array or an `NI_DAQ_Mem` array. **buffer** must have a length not less than **count**. When `Lab_ISCAN_Op` returns with an error code of zero, **buffer** contains the acquired data.

Lab_ISCAN_Op

Continued

count is the number of samples to be acquired (that is, the number of A/D conversions to be performed).

Range: 3 through 2^{32} - 1 (except Lab and 1200 series devices, which are limited to 65,535).

sampleRate is the sample rate you want in units of pts/s.

Range: Roughly 0.00153 pts/s through 62,500 pts/s (Lab and 1200 series devices). Roughly 0.00153 pts/s through 50,000 pts/s (DAQCard-500/700 and 516 and LPM devices).

Note: ***If you are using an SCXI-1200 with remote SCXI, the maximum rate will depend on the baud rate setting and* count*. Refer to the* SCXI-1200 User Manual *for more details.***

scanRate is the scan rate you want in units of scans/s. This is the rate at which NI-DAQ performs scans. NI-DAQ performs a scan each time NI-DAQ samples all channels in the scan sequence. S**canRate** must be slightly less than **sampleRate/numChans** due to a 5 µs delay interval to the driver. `Lab_ISCAN` interval scanning is available on the Lab and 1200 series devices only.

Range: 0 and roughly 0.00153 scans/s through 62,500 scans/s.

Note: ***If you are using an SCXI-1200 with remote SCXI, the maximum rate will depend on the baud rate setting and* count*. Refer to the* SCXI-1200 User Manual *for more details.***

A value of 0 disables interval scanning.

finalScanOrder is an array that indicates the scan channel order of the data in the buffer passed to `Lab_ISCAN_Op`. The size of **finalScanOrder** must be at least equal to the number of channels scanned. This parameter is valid only when the error is returned to zero and is useful only when pretrigger mode is enabled (Lab and 1200 series devices only).

If you do not use pretrigger mode, the values contained in **finalScanOrder** are, in single-ended mode, *n*-1, *n*-2, ..., 1, 0, in that order, and in differential mode, 2 (*n*-1), 2(*n*-2), ..., 1, 0, in that order, where *n* is the number of channels scanned. For example, if you scanned three channels in single-ended mode, the **finalScanOrder** returns:

finalScanOrder[0] = 2.
finalScanOrder[1] = 1.
finalScanOrder[2] = 0.

Lab_ISCAN_Op

Continued

So the first sample in the buffer belongs to channel 2, the second sample belongs to channel 1, the third sample belongs to channel 0, the fourth sample belongs to channel 2, and so on. This is exactly the scan order you would expect from the Lab and 1200 series devices and **finalScanOrder** is not useful in this case.

If you use pretrigger mode, the order of the channel numbers in **finalScanOrder** depends on where in the scan sequence the acquisition ended. This can vary because the stop trigger can occur in the middle of a scan sequence, which causes the acquisition to end in the middle of a scan sequence so that the oldest data point in the buffer can belong to any channel in the scan sequence. `Lab_ISCAN_Op` rearranges the buffer so that the oldest data point is at index 0 in the buffer. This rearrangement causes the scanning order to change. This new scanning order is returned by **finalScanOrder**. For example, if you scanned three channels, the original scan order is channel 2, channel 1, channel 0, channel 2, and so on. However, after the stop trigger, if the acquisition ends after taking a sample from channel 1, the oldest data point belongs to channel 0.

So **finalScanOrder** returns:
finalScanOrder[0] = 0.
finalScanOrder[1] = 2.
finalScanOrder[2] = 1.

Therefore the first sample in the buffer belongs to channel 0, the second sample belongs to channel 2, the third sample belongs to channel 1, the fourth sample belongs to channel 0, and so on.

Using This Function

`Lab_ISCAN_Op` initiates a synchronous process of acquiring A/D conversion samples and storing them in a buffer. `Lab_ISCAN_Op` does not return control to your application until NI-DAQ acquires all the samples you want (or until an acquisition error occurs). When you use posttrigger mode, the process stores **count** A/D conversions in the buffer and ignores any subsequent conversions.

☞ **Note:** ***If you have selected external start triggering of the data acquisition operation, a low-to-high edge at the EXTTRIG of the Lab and 1200 series device I/O connector input initiates the operation. Be aware that if you do not apply the start trigger,*** `Lab_ISCAN_Op` ***does not return control to your application. Otherwise,*** `Lab_ISCAN_Op` ***issues a software trigger to initiate the data acquisition operation.***

Lab_ISCAN_Op

Continued

If you have enabled pretrigger mode, the sample counter does not begin counting acquisitions until you apply a signal at the stop trigger input. Until you apply this signal, the acquisition remains in a cyclical mode, continually overwriting old data in the buffer with new data. Again, if the stop trigger is not applied, `Lab_ISCAN_Op` does not return control to your application.

In any case, you can use `Timeout_Config` to establish a maximum length of time for `Lab_ISCAN_Op` to execute.

Lab_ISCAN_Start

Format

status = Lab_ISCAN_Start (deviceNumber, numChans, gain, buffer, count, sampTimebase, sampInterval, scanInterval)

Purpose

Initiates a multiple-channel scanned data acquisition operation and stores its input in an array (DAQCard-500/700 and 516, Lab and 1200 series, and LPM devices only).

Parameters

Input

Name	Type		Description
	Windows	Windows NT	
deviceNumber	I16	I32	assigned by configuration utility
numChans	I16	I32	number of channels to be scanned
gain	I16	I32	gain setting
count	U32	U32	total number of samples to be acquired
sampTimebase	I16	I32	timebase, or resolution, used for the sample-interval counter
sampInterval	U16	U32	length of the sample interval
scanInterval	U16	U32	length of the scan interval

Lab_ISCAN_Start

Continued

Output

Name	Type		Description
	Windows	Windows NT	
buffer	[I16], HDL	[I16], HDL	results of the scanned data acquisition

Parameter Discussion

numChans is the number of channels to be scanned in a single scan sequence. The value of this parameter also determines which channels NI-DAQ scans because these supported devices have a fixed scanning order. The scanned channels range from **numChans** - 1 to channel 0. If you are using SCXI modules with additional multiplexers, you must scan the appropriate analog input channels on the DAQ device that corresponds to the SCXI channels you want. You should select the SCXI scan list using `SCXI_SCAN_Setup` before you call this function. Please refer to Chapter 15, *SCXI Hardware*, in the *DAQ Hardware Overview Guide* and the *NI-DAQ User Manual for PC Compatibles* for more information on SCXI channel assignments.

Range: 1 through 4 for the 516 and Lab and 1200 series devices in differential mode.
1 through 8 for the DAQCard-500 (single-ended mode only).
1 through 8 for the DAQCard-700 in differential mode.
1 through 8 for the 516 and Lab and 1200 series devices in single-ended mode.
1 through 16 for the DAQCard-700 and LPM devices in single-ended mode.

gain is the gain setting to be used for the scanning operation. NI-DAQ applies the same gain to all the channels scanned. This gain setting applies only to the DAQ device; if you are using SCXI modules with additional gain selection, you must establish any gain you want at the SCXI module either by setting jumpers on the module or by calling `SCXI_Set_Gain`. The following gain settings are valid for the Lab and 1200 series devices: 1, 2, 5, 10, 20, 50, 100. If you use an invalid gain setting, NI-DAQ returns an error. NI-DAQ ignores **gain** for the DAQCard-500/700 and 516 and LPM devices.

buffer is an integer array or an `NI_DAQ_Mem` array. **buffer** must have a length equal to or greater than **count**.

count is the total number of samples to be acquired (that is, the number of A/D conversions to be performed). For double-buffered acquisitions, **count** must be even.

Lab_ISCAN_Start

Continued

Range: 3 through 2^{32} - 1 (except the Lab and 1200 series devices, which are limited to 65,535 unless enabled for double-buffered mode).

sampTimebase is the timebase, or resolution, to be used for the sample-interval counter. The sample-interval counter controls the time that elapses between acquisition of samples within a scan sequence.

sampTimebase has the following possible values:

1:	1 MHz clock used as timebase (1 µs resolution).
2:	100 kHz clock used as timebase (10 µs resolution).
3:	10 kHz clock used as timebase (100 µs resolution).
4:	1 kHz clock used as timebase (1 ms resolution).
5:	100 Hz clock used as timebase (10 ms resolution).

If sample-interval timing is to be externally controlled, NI-DAQ ignores **sampTimebase** and the parameter can be any value.

sampInterval indicates the length of the sample interval (that is, the amount of time to elapse between each A/D conversion within a scan sequence).
Range: 2 through 65,535.

The sample interval is a function of the timebase resolution. NI-DAQ determines the actual sample interval in seconds by the following formula:

sampInterval ∗ (sample timebase resolution)

where the sample timebase resolution is equal to one of the values of **sampTimebase** as specified above. For example, if **sampInterval** = 25 and **sampTimebase** = 2, the actual sample interval is 25 ∗ 10 µs = 250 µs. The total sample interval (the time to complete one scan sequence) in seconds is the actual sample interval ∗ number of channels scanned. If the sample interval is to be externally controlled by conversion pulses applied to the EXTCONV* input, NI-DAQ ignores the **sampInterval** and the parameter can be any value.

scanInterval indicates the length of the scan interval. This is the amount of time to elapse between scans. The function performs a scan each time NI-DAQ samples all channels in the scan sequence. Therefore, **scanInterval** must be greater than or equal to **sampInterval** * **numChans** +5 µs.
Range: 0 and 2 through 65,535.
A value of 0 disables interval scanning. `Lab_ISCAN` interval scanning is not available on the DAQCard-500/700 and 516 and LPM devices.

Lab_ISCAN_Start

Continued

Using This Function

If you did not specify external sample-interval timing by the `DAQ_Config` call, NI-DAQ sets the sample-interval counter to the specified **sampInterval** and **sampTimebase**, and sets the sample counter up to count the number of samples acquired and to stop the data acquisition process when the number of samples acquired equals **count**. If you have specified external sample-interval timing, the data acquisition circuitry relies on pulses received on the EXTCONV* input to initiate individual A/D conversions.

`Lab_ISCAN_Start` initializes a background data acquisition process to handle storing of A/D conversion samples into the buffer as NI-DAQ acquires them. When you use posttrigger mode (with pretrigger mode disabled), the process stores up to **count** A/D conversion samples into the buffer and ignores any subsequent conversions. The order of the scan is from channel *n-1* to channel 0, where *n* is the number of channels being scanned. For example, if **numChans** is 3 (that is, you are scanning three channels), NI-DAQ stores the data in the buffer in the following order:

> First sample from channel 2, first sample from channel 1, first sample from channel 0, second sample from channel 2, and so on.

You cannot make the second call to `Lab_ISCAN_Start` without terminating this background data acquisition process. If a call to `Lab_ISCAN_Check` returns **daqStopped** = 1, the samples are available and NI-DAQ terminates the process. In addition, a call to `DAQ_Clear` terminates the background data acquisition process. Notice that if a call to `Lab_ISCAN_Check` returns an error code of -75 or -76, or **daqStopped** = 1, the process is automatically terminated and there is no need to call `DAQ_Clear`.

For the Lab and 1200 series devices, if you enable pretrigger mode, `Lab_ISCAN_Start` initiates a cyclical acquisition that continually fills the buffer with data, wrapping around to the start of the buffer once NI-DAQ has written to the entire buffer. When you apply the signal at the stop trigger input, `Lab_ISCAN_Start` acquires an additional number of samples specified by the **ptsAfterStoptrig** parameter in `DAQ_StopTrigger_Config` and then terminates.

Since the trigger can occur at any point in the scan sequence, the scanning operation can end in the middle of a scan sequence. See the description for `Lab_ISCAN_Check` to determine how NI-DAQ rearranges the buffer after the acquisition ends. When you enable pretrigger mode, the length of the buffer, which is greater than or equal to **count**, should be an integral multiple of **numChans**.

Lab_ISCAN_Start

Continued

If you have selected external start triggering of the data acquisition operation, a low-to-high edge at the EXTTRIG of the Lab and 1200 series device I/O connector input initiates the data acquisition operation after the `Lab_ISCAN_Start` call is complete. Otherwise, `Lab_ISCAN_Start` issues a software trigger to initiate the data acquisition operation before returning.

☞ **Note:** ***If your application calls*** `Lab_ISCAN_Start`***, always ensure that you call*** `DAQ_Clear` ***before your application terminates and returns control to the operating system. Unless you make this call (either directly, or indirectly through*** `Lab_ISCAN_Check` ***or*** `DAQ_DB_Transfer`***), unpredictable behavior may result.***

Lab_ISCAN_to_Disk

Format

status = Lab_ISCAN_to_Disk (deviceNumber, numChans, gain, filename, count, sampleRate, scanRate, concat)

Purpose

Performs a synchronous, multiple-channel scanned data acquisition operation and simultaneously saves the acquired data in a disk file. `Lab_ISCAN_to_Disk` does not return until NI-DAQ has acquired and saved all the data or an acquisition error has occurred (DAQCard-500/700 and 516, Lab and 1200 series, and LPM devices only).

Parameters

Input

Name	Type		Description
	Windows	Windows NT	
deviceNumber	I16	I32	assigned by configuration utility
numChans	I16	I32	number of channels to be scanned
gain	I16	I32	gain setting
filename	STR	STR	name of the data file to be created
count	U32	U32	number of samples to be acquired
sampleRate	F64	F64	desired sample rate in units of pts/s

Lab_ISCAN_to_Disk

Continued

Name	Type		Description
	Windows	Windows NT	
scanRate	F64	F64	desired scan rate in units of pts/s
concat	I16	I32	enables concatenation of data to an existing file

Parameter Discussion

numChans is the number of channels to be scanned in a single scan sequence. The value of this parameter also determines which channels NI-DAQ scans because these supported devices have a fixed scanning order. The scanned channels range from **numChans** - 1 to channel 0. If you are using SCXI modules with additional multiplexers, you must scan the appropriate analog input channels on the DAQ device that corresponds to the SCXI channels you want. You should select the SCXI scan list using `SCXI_SCAN_Setup` before you call this function. Please refer to Chapter 15, *SCXI Hardware*, in the *DAQ Hardware Overview Guide* and the *NI-DAQ User Manual for PC Compatibles* for more information on SCXI channel assignments.

Range:	1 through 4 for the 516 and Lab and 1200 series devices in differential mode.
	1 through 8 for the DAQCard-500 (single-ended mode only).
	1 through 8 for the DAQCard-700 in differential mode.
	1 through 8 for the 516 and Lab and 1200 series devices in single-ended mode.
	1 through 16 for the DAQCard-700 and LPM devices in single-ended mode.

gain is the gain setting to be used for the scanning operation. NI-DAQ applies the same gain to all the channels scanned. This gain setting applies only to the DAQ device; if you use SCXI modules with additional gain selection, you must establish any gain you want at the SCXI module either by setting jumpers on the module or by calling `SCXI_Set_Gain`. The following gain settings are valid for the Lab and 1200 series devices: 1, 2, 5, 10, 20, 50, 100. If you use an invalid gain setting, NI-DAQ returns an error. NI-DAQ ignores **gain** for the DAQCard-500/700 and LPM devices.

count is the number of samples to be acquired (that is, the number of A/D conversions to be performed). The length of your data file should be exactly twice the value of **count**. If you have previously enabled pretrigger mode (by a call to `DAQ_StopTrigger_Config`) NI-DAQ ignores the **count** parameter.

Lab_ISCAN_to_Disk

Continued

Range: 3 through 2^{32} - 1.

sampleRate is the sample rate you want in units of pts/s.
Range: Roughly 0.00153 pts/s through 62,500 pts/s (Lab and 1200 series devices). Roughly 0.00153 pts/s through 50,000 pts/s (DAQCard-500/700 and 516 and LPM devices).

☞ **Note:** ***If you are using an SCXI-1200 with remote SCXI, the maximum rate will depend on the baud rate setting and*** **count*****. Refer to the*** *SCXI-1200 User Manual* ***for more details.***

scanRate is the scan rate you want in units of pts/s. This is the rate at which NI-DAQ performs scans. The function performs a scan each time NI-DAQ samples all channels in the scan sequence. Therefore, **scanRate** must be equal to or greater than **sampleRate** * **numChans**. `Lab_ISCAN` interval scanning is available on the Lab-PC+ and SCXI-1200 only.
Range: 0 and roughly 0.00153 pts/s through 62,500 pts/s.

☞ **Note:** ***If you are using an SCXI-1200 with remote SCXI, the maximum rate will depend on the baud setting. Refer to the*** *SCXI-1200 User Manual* ***for more details.***

A value of 0 disables interval scanning.

concat enables concatenation of data to an existing file. Regardless of the value of **concat**, if the file does not exist, NI-DAQ creates the file.

0:	Overwrite file if it exists.
1:	Concatenate new data to an existing file.

Using This Function

`Lab_ISCAN_to_Disk` initiates a synchronous process of acquiring A/D conversion samples and storing them in a disk file. `Lab_ISCAN_to_Disk` does not return control to your application until NI-DAQ acquires and saves all the samples you want (or until an acquisition error occurs). For the Lab and 1200 series devices, when you use posttrigger mode, the process stores **count** A/D conversions in the file and ignores any subsequent conversions.

☞ **Note:** ***If you have selected external start triggering of the data acquisition operation, a low-to-high edge at the EXTTRIG of the Lab and 1200 series device I/O connector input initiates the data acquisition operation. Be***

Lab_ISCAN_to_Disk

Continued

aware that if you do not apply the start trigger, `Lab_ISCAN_to_Disk` ***does not return control to your application. Otherwise,*** `Lab_ISCAN_to_Disk` ***issues a software trigger to initiate the data acquisition operation.***

If you have enabled pretrigger mode, the sample counter does not begin counting acquisitions until you apply a signal at the stop trigger input. Until you apply this signal, the acquisition continues to write data into the disk file. NI-DAQ ignores the value of the **count** parameter when you enable pretrigger mode. If you do not apply the stop trigger, `Lab_ISCAN_to_Disk` eventually returns control to your application because, sooner or later, you will run out of disk space.

In any case, you can use `Timeout_Config` to establish a maximum length of time for `Lab_ISCAN_to_Disk` to execute.

LPM16_Calibrate

Format

status = LPM16_Calibrate (deviceNumber)

Purpose

Calibrates the LPM devices converter. The calibration calculates the correct offset voltage for the voltage comparator, adjusts positive linearity and full-scale errors to less than ±0.5 LSB each, and adjusts zero error to less than ±1 LSB.

Parameters

Input

Name	**Type**		**Description**
	Windows	**Windows NT**	
deviceNumber	I16	I32	assigned by configuration utility

Using This Function

When the function is called, the ADC1241 ADC goes into a self-calibration cycle. The function does not return until the self-calibration is completed.

MAI_Arm

Format

status = MAI_Arm (deviceNumber, armDisarm)

Purpose

Enables/disables the device to take a sample of selected input channels whenever the device receives an external pulse on the SAMPCLK* input or the CLOCKI RTSI bus input. The data is stored in the A/D FIFO of the device for later retrieval by `MAI_Read` (EISA-A2000 only).

Parameters

Input

Name	Type		Description
	Windows	Windows NT	
deviceNumber	I16	I32	assigned by configuration utility
armDisarm	I16	I32	whether or not to arm the device

Parameter Discussion

armDisarm indicates whether or not to arm the device.

1:	Arm the device to convert on external conversion pulses.
0:	Disarm the device to ignore external conversion pulses.

Using This Function

When you call `MAI_Arm` with **armDisarm** set to 1, NI-DAQ clears the A/D FIFO and configures the device to sample the input signals whenever the device receives a falling edge on the SAMPCLK* input on the EISA-A2000 I/O connector or the device receives a rising edge on the CLOCKI RTSI bus input. To retrieve these values, call `MAI_Read`. When you call `MAI_Arm` with **armDisarm** set to 0, the device ignores any subsequent pulses received on the external sample clock line. NI-DAQ does not clear the A/D FIFO when NI-DAQ disarms the device.

MAI_Arm

Continued

You must call `A2000_Config` to select the external sample clock before you call `MAI_Arm` to arm the device. Otherwise, `MAI_Arm` returns a **multClkSrcError**. You must call `RTSI_Conn` if you use the CLOCKI RTSI bus input (see the *RTSI Bus Trigger Functions* section of Chapter 3, *Software Overview*, of the *NI-DAQ User Manual for PC Compatibles*). After calling `MAI_Arm` to disarm, be sure to call `RTSI_DisConn` or `RTSI_Clear`, if necessary, and `A2000_Config` to reset to the internal sample clock.

MAI_Clear

Format

status = MAI_Clear (deviceNumber)

Purpose

Clears the A/D FIFO and related analog input circuitry.

Parameters

Input

Name	**Type**		**Description**
	Windows	**Windows NT**	
deviceNumber	I16	I32	assigned by configuration utility

Using This Function

Call `MAI_Clear` to clear any analog input error conditions and unwanted samples from the A/D FIFO.

MAI_Coupling

Format

status = MAI_Coupling (deviceNumber, numChans, coupling)

Purpose

Selects AC or DC coupling for all channels with programmable AC/DC coupling.

Parameters

Input

Name	Type		Description
	Windows	Windows NT	
deviceNumber	I16	I32	assigned by configuration utility
numChans	I16	I32	number of channels that have programmable coupling settings
coupling	[I16]	[I32]	selects AC or DC coupling for each analog input channel

Parameter Discussion

numChans is the number of channels on the device that have programmable coupling settings.

5:	EISA-A2000 channels ACH0, ACH1, ACH2, ACH3, and ATRIG.
4:	AT-A2150 channels ACH0, ACH1, ACH2, and ACH3.
2:	AT-DSP2200 channels ACH0 and ACH1.

coupling is an array of length **numChans** that selects AC or DC coupling for each analog input channel. Each value in the **coupling** array selects the setting for each channel as follows:

coupling[*i*] = 0: AC coupling.
coupling[*i*] = 1: DC coupling.

MAI_Coupling

Continued

For the EISA-A2000, the elements in **coupling** are interpreted as follows:
coupling[0]—coupling setting for ACH0.
coupling[1]—coupling setting for ACH1.
coupling[2]—coupling setting for ACH2.
coupling[3]—coupling setting for ACH3.
coupling[4]—coupling setting for ATRIG.

For the AT-A2150, the elements in **coupling** are interpreted as follows:
coupling[0]—coupling setting for ACH0.
coupling[1]—coupling setting for ACH1.
coupling[2]—coupling setting for ACH2.
coupling[3]—coupling setting for ACH3.

For the AT-DSP2200, the elements in **coupling** are interpreted as follows:
coupling[0]—coupling setting for ACH0.
coupling[1]—coupling setting for ACH1.

☞ **Note:** ***All programmable channels must have a setting in the* coupling *array when you call* `MAI_Coupling`*.***

Using This Function

`MAI_Coupling` sets each analog input to the selected coupling. After system startup or an `Init_DA_Brds` function call, the coupling of all programmable channels defaults to AC coupling for an EISA-A2000 and DC coupling for an AT-A2150 and AT-DSP2200.

MAI_Read

Format

status = MAI_Read (deviceNumber, reading)

Purpose

Returns a reading for all of the selected analog input channels. If you are using an external sample clock and you have called `MAI_Arm`, samples generated by previous sample clock pulses are returned; otherwise, NI-DAQ clears the A/D FIFO, generates an A/D conversion pulse, and returns the samples produced by this pulse.

Parameters

Input

Name	Type		Description
	Windows	Windows NT	
deviceNumber	I16	I32	assigned by configuration utility

Output

Name	Type		Description
	Windows	Windows NT	
reading	[I16]	[I16]	readings from each sampled analog input channel

Parameter Discussion

reading is an array of readings from each sampled analog input channel. The length of the **reading** array is equal to the number of channels selected in the `MAI_Setup` **numChans** parameter. On the EISA-A2000, the elements of **reading** are 12-bit sign extended integers which range from -2,048 to +2,047. On the AT-A2150 and AT-DSP2200, the elements of **reading** are 16-bit signed integers which range from -32,768 to +32,767.

MAI_Read

Continued

Using This Function

`MAI_Read` samples the analog input channels selected by `MAI_Setup`. On an EISA-A2000 or AT-A2150 by default, NI-DAQ samples all four analog input channels. On an AT-DSP2200 by default, NI-DAQ samples both analog input channels.

☞ **Note:** ***EISA-A2000 only—The default state of the EISA-A2000 is AC coupling. If you are always reading zero when reading a DC voltage on your EISA-A2000, the board may be in AC coupling mode. See the*** `MAI_Coupling` ***function for more information.***

If you use the external sample clock and call `MAI_Arm`, `MAI_Read` returns the earliest reading stored in the A/D FIFO for the selected channels. If you use the external sample clock and have not called `MAI_Arm`, `MAI_Read` returns a **notArmedError**. NI-DAQ returns a **timeOutError** if no readings are stored. You can use `MAI_Clear` at any time to clear the A/D FIFO and an overflow error condition.

If you are using an AT-DSP2200, please note that `MAI_Read` does not support DSP memory. That is, the **reading** array must always be an array in PC memory. Also note that the translation option is not supported (see `DSP2200_Config`). That is, the **reading** array must always be an array of 16-bit integers.

MAI_Scale

Format

status = MAI_Scale (deviceNumber, numScans, values, volts)

Purpose

Given an array of acquired data, converts the values in the array to the actual voltage values measured.

Parameters

Input

Name	Type		Description
	Windows	Windows NT	
deviceNumber	I16	I32	assigned by configuration utility
numScans	U32	U32	total number of scans in the values array
values	[I16]	[I16]	acquired binary data

Output

Name	Type		Description
	Windows	Windows NT	
volts	[F64]	[F64]	double-precision values

Parameter Discussion

numScans is the total number of scans in the values array. A scan can consist of 1, 2, or 4 samples depending on the value of **numChans** when you last called `MAI_Setup`.

values is an array of acquired 12-bit or 16-bit binary data depending on the device type.

MAI_Scale

Continued

volts is an array of double-precision values that correspond to the actual input voltages measured.

Using This Function

`MAI_Scale` calculates **volts** from **values** as follows:
For the EISA-A2000: **volts**[*i*] = **values**[*i*] ∗ (5 / 2,048).
For the AT-A2150 and AT-DSP2200: **volts**[*i*] = **values**[*i*] ∗ (2.828 / 32,768).

☞ **Note:** `MAI_Scale` ***assumes that the value of*** **numChans** ***that was in effect during the data acquisition is the same value as the value of numChans that is in effect when NI-DAQ is scaling the data. If NI-DAQ logs data to disk and then reads the data later for scaling, you need to call the*** `MAI_Setup` ***function before scaling to set*** **numChans** ***to its correct value.***

MAI_Setup

Format

status = MAI_Setup (deviceNumber, numChans, chanList, gainList, muxInterval, timebase, muxMode)

Purpose

Selects the analog input channels read, sets the gain per channel, and sets the multiplexing rate between channels for all analog input operations. `MAI_Setup` affects single-read, multiple-channel analog input (`MAI`), and multiple-channel data acquisition (`MDAQ`) operations.

Parameters

Input

Name	Type		Description
	Windows	Windows NT	
deviceNumber	I16	I32	assigned by configuration utility
numChans	I16	I32	number of onboard channels
chanList	[I16]	[I32]	listing of the analog input channels to be scanned
gainList	[I16]	[I32]	contains a gain setting for each channel selected
muxInterval	U16	U32	input multiplexer switching time interval
timebase	I16	I32	resolution to use for the multiplexer switching interval
muxMode	I16	I32	number of external multiplexer devices connected

Parameter Discussion

numChans is the number of onboard channels to be scanned when sampling the analog input. Values of 1, 2, and 4 are valid for the EISA-A2000 and AT-A2150. Values of 1 and 2 are valid when using an AT-DSP2200.

chanList is an integer array of length **numChans** that contains a listing of the analog input channels to be scanned. Table 2-2 outlines the valid combinations and ordering of channels for the EISA-A2000, AT-A2150, and AT-DSP2200.

Table 2-6. Valid numChans and chanList Settings for the EISA-A2000, AT-A2150, and AT-DSP2200

numChans	chanList Entries	
1	**chanList**[0] = 0 or 1 or 2 or 3	samples a single analog input
2	**chanList**[0] = 0, **chanList**[1] = 1, or **chanList**[0] = 2, **chanList**[1] = 3	samples two analog inputs
4	**chanList**[0] = 0, **chanList**[1] = 1, **chanList**[2] = 2, **chanList**[3] = 3	samples four analog inputs (EISA-A2000 and AT-A2150 only)

gainList is an integer array of length **numChans** that contains a gain setting for each channel selected in **chanList**. The EISA-A2000, AT-A2150, and AT-DSP2200 have a fixed gain of 1; therefore, NI-DAQ ignores the **gainList** value.

muxInterval is the input multiplexer switching time interval, that is, the time lapse between the sampling of each successive channel in **chanList**. The multiplexer switching time interval is a function of the timebase resolution selected by timebase. NI-DAQ determines the actual interval in seconds using the following formula:

muxInterval $*$ (timebase resolution)

Because the EISA-A2000, AT-A2150, and AT-DSP2200 simultaneously sample their input, **muxInterval** is always zero and is ignored by NI-DAQ.

timebase is the resolution to use for the multiplexer switching interval. Because **muxInterval** is always zero for the EISA-A2000, AT-A2150, and AT-DSP2200, NI-DAQ ignores **timebase**.

MAI_Setup

Continued

muxMode indicates the number of external multiplexer devices connected to the device. Because the EISA-A2000, AT-A2150, and AT-DSP2200 do not use an external multiplexer device, NI-DAQ ignores **muxMode**.

Using This Function

The EISA-A2000, AT-A2150, and AT-DSP2200 are initially configured at system startup to sample all their input channels. `MAI_Setup` is needed only if you want to sample fewer channels or if you have changed the configured input channels.

Master_Slave_Config

Format

status = Master_Slave_Config (deviceNumber, numSlaves, slaveList)

Purpose

Configures one device as a master device and one or more remaining devices as slave devices. This function ensures that, in a multiple-frame acquisition, the slave devices are always re-enabled *before* the master device.

Parameters

Input

Name	Type		Description
	Windows	Windows NT	
deviceNumber	I16	I32	assigned by configuration utility
numSlaves	I16	I32	number of slave devices to be controlled
slaveList	[I16]	[I32]	contains the device (slot) numbers of the slave devices

Parameter Discussion

numSlaves indicates the number of slave devices to be controlled in some way by the master device.

0:	Erases the master/slave relationship that you previously established between the master device (selected by the value of the **deviceNumber** parameter) and its slave devices.
1-7:	The master is controlling from one to seven slave devices.

slaveList is an array of length **numSlaves** that contains the device (slot) numbers of the slave devices.

Range: 1-8.

Master_Slave_Config

Continued

Using This Function

Currently, this function is used by the EISA-A2000, AT-A2150, and AT-DSP2200 when the device is writing to computer memory (not onboard memory). A device is considered a master device if it is sending a trigger or clock signal to another device (see the *RTSI Bus Trigger Functions* section of Chapter 3, *Software Overview*, of the *NI-DAQ User Manual for PC Compatibles* for more information on these signals). The device receiving these signals is considered a slave device because its sampling is in some way controlled by signals sent from the master device. In a multiple-frame acquisition, for a slave device to always be able to respond to a master signal, you must enable the slave device before the master device. If you enable the master device first, it can send its signal to the slave devices before they are capable of responding. The initial startup order is the responsibility of your application. You should always start the master device last. The purpose of `Master_Slave_Config` is to ensure that NI-DAQ will arm the master device last for each subsequent frame acquired during a multiple-frame acquisition.

MDAQ_Check

Format

status = MDAQ_Check (deviceNumber, fullCheck, acqDone, framesDone, scansDone)

Purpose

Reports the current status of the data acquisition—whether the acquisition is complete—and the number of acquired frames and scans.

Parameters

Input

Name	Type		Description
	Windows	Windows NT	
deviceNumber	I16	I32	assigned by configuration utility
fullCheck	I16	I32	whether to return complete or partial status information

Output

Name	Type		Description
	Windows	Windows NT	
acqDone	I16	I32	whether the data acquisition has completed
framesDone	U32	U32	the number of the last completed frame
scansDone	U32	U32	the most recent scan number that has been acquired within the current frame

MDAQ_Check

Continued

Parameter Discussion

fullCheck indicates whether to return complete or partial status information. Partial information includes all parameters except **scansDone**.

0:	Returns partial status information.
1:	Returns complete status information.

Updating the **scansDone** parameter requires temporarily suspending servicing of the device being checked. If you are performing many **fullChecks** during a high-speed acquisition or while you are operating many devices together, a device FIFO may overflow and halt the data acquisition on that device. For these cases and when you want only the status information concerning the completion of the acquisition, setting **fullCheck** to 0 is recommended.

acqDone returns an indication of whether the data acquisition has completed.

0:	The data acquisition is not yet complete.
1:	The acquisition is complete.

If you have chosen either continuous acquisition of scans in the `MDAQ_Setup` function or unlimited acquisition of frames after the first trigger in the `MDAQ_Start` function, the status remains zero until `MDAQ_Stop` or an error (such as a FIFO overflow) has stopped the acquisition, after which **acqDone** becomes 1. Even though the AT-DSP2200 does not have a FIFO, NI-DAQ still uses the FIFO overflow error (-75) for this device to indicate that data has been lost.

framesDone returns the number of the last completed frame. In frame-oriented acquisitions, **framesDone** ranges from zero (when the acquisition has not yet started—for example, when waiting for the first trigger in posttrigger mode) to the total number of frames to be acquired (as defined in `MDAQ_Start`). In scan-oriented acquisitions, **framesDone** is 0 until the acquisition is complete, at which time **framesDone** becomes 1.

scansDone returns the most recent scan number that NI-DAQ has acquired within the current frame. In posttrigger frame-oriented acquisitions, **scansDone** ranges from zero (when the frame has not yet started—for example, when waiting for a trigger in posttrigger mode) to the number of scans in a frame (as defined in `MDAQ_Setup`). If the frames include pretrigger scans, **scansDone** returns all the scans NI-DAQ acquires in a frame until the trigger arrives and NI-DAQ acquires the number of posttrigger scans you want. After this point, if no more frames are to be acquired (**acqDone** equals 1), **scansDone** returns a number equal to the number of scans in a frame. In scan-oriented

MDAQ_Check

Continued

acquisitions, **scansDone** ranges from zero to the number of posttrigger scans to collect. When **fullCheck** is 0, **scansDone** is always set to 0.

☞ **Note:** ***C Programmers*—acqDone, framesDone, *and* scansDone *are pass-by-reference parameters.***

MDAQ_Clear

Format

status = MDAQ_Clear (deviceNumber)

Purpose

Stops data acquisition and releases internal resources so that your application can successfully terminate.

Parameters

Input

Name	Type		Description
	Windows	Windows NT	
deviceNumber	I16	I32	assigned by configuration utility

Using This Function

`MDAQ_Clear` stops the acquisition process and prepares the system for the termination of the application. NI-DAQ releases certain internal buffers and system resources. One result is that the `MDAQ_Get` call is unable to *unwrap* pretrigger data.

Although the current hardware settings remain in effect after calling `MDAQ_Clear`, you must call `MDAQ_Setup` again before you can make another `MDAQ_Start` call.

You must call `MDAQ_Clear` before exiting your application; otherwise, resources are not properly deallocated.

MDAQ_Get

Format

status = MDAQ_Get (deviceNumber, scansOrFrames, getOrTap, numToGet, startFrame, startScan, timeout, getBuffer, numGotten, lastFrame, lastScan, acqDone)

Purpose

Transfers acquired data from the buffer into the buffer you specified while data acquisition is in progress or after data acquisition is complete. `MDAQ_Get` can retrieve data from anywhere in the acquisition buffer.

Parameters

Input

Name	Type		Description
	Windows	Windows NT	
deviceNumber	I16	I32	assigned by configuration utility
scansOrFrames	I16	I32	retrieve scans or frames
getOrTap	I16	I32	the retrieval method used
numToGet	U32	U32	number of scans or frames to retrieve
startFrame	U32	U32	frame number to begin
startScan	U32	U32	scan number to begin
timeout	I32	I32	number of clock ticks to wait for data

MDAQ_Get

Continued

Output

Name	Type		Description
	Windows	Windows NT	
getBuffer	[I16], [F32], HDL	[I16], [F32], HDL	memory space to store the data retrieved
numGotten	U32	U32	the number of items actually copied
lastFrame	U32	U32	returns the number of the last frame
lastScan	U32	U32	returns the last scan number
acqDone	I16	I32	whether the data acquisition has completed

Parameter Discussion

deviceNumber is the assigned by configuration utility of the DAQ device.

scansOrFrames indicates whether to retrieve scans or frames from the acquisition **getBuffer**. NI-DAQ can retrieve scans from either a scan-oriented or a frame-oriented acquisition (see `MDAQ_Setup`). NI-DAQ can retrieve frames only from a frame-oriented acquisition.

0: Get scans.
1: Get frames.

getOrTap indicates the retrieval method used to retrieve data from the acquisition buffer.

0: (performing a *GET*)—Get oldest data sequentially.
1: (performing a *TAP*)—Get most recently acquired data.

In either case, indicate a starting frame and scan when you can. When performing a *GET*, NI-DAQ uses two internal sequential retrieval pointers (one for the starting frame and one for the starting scan) to keep track of what the function has retrieved to guarantee sequential access. NI-DAQ maintains no such pointers when performing a *TAP*.

MDAQ_Get

Continued

numToGet is the number of scans or frames to retrieve from the acquisition buffer. If **scansOrFrames** is 0 (retrieving scans) and the acquisition is frame oriented, **numToGet** ranges from 1 to the number of scans in a frame (**preTrigScans** + **postTrigScans**). NI-DAQ cannot retrieve scans across frame boundaries. Always check the **numGotten** parameter to determine the actual number of scans NI-DAQ retrieved. If **scansOrFrames** is 1, **numToGet** ranges from 1 to the number of frames in the acquisition buffer. The **numGotten** parameter indicates the number of frames NI-DAQ actually retrieved.

startFrame is the frame number to begin copying from. If the acquisition is scan oriented, NI-DAQ ignores **startFrame** so the following cases are only relevant for frame-oriented acquisitions. If performing a *GET* (**getOrTap** = 0) and **startFrame** is not 0, copying begins in **startFrame**. If performing a *GET* and **startFrame** is 0, NI-DAQ uses the retrieval frame pointer to determine where to begin copying. In both cases, after retrieving the data, NI-DAQ resets the retrieval frame pointer to the frame immediately following the last frame copied. If performing a *TAP* (**getOrTap** = 1) and **startFrame** is not 0, copying begins in **startFrame**. If performing a *TAP* and **startFrame** is 0, NI-DAQ copies the most recently acquired data. When performing a *TAP*, NI-DAQ does not use or modify internal retrieval pointers.

startScan is the scan number to begin copying from within **startFrame**. If **scansOrFrames** is 1 (retrieving frames), NI-DAQ ignores **startScan**. For the following cases, **scansOrFrames** is 0 (retrieving scans). If performing a *GET* and **startScan** is not 0, copying begins at **startScan** (**startFrame** must also be not 0 or NI-DAQ returns an error). If performing a *GET* and **startScan** is 0, copying begins at the scan pointed to by the retrieval scan pointer (**startFrame** must also be 0 or NI-DAQ returns an error). In both cases, after retrieving the data, NI-DAQ resets the retrieval scan pointer to the scan immediately following the last scan copied. If performing a *TAP* and **startScan** is not 0, copying begins at **startScan**. If performing a *TAP* and **startScan** is 0, NI-DAQ copies the most recently acquired data. When performing a *TAP*, any combination of valid values for **startScan** and **startFrame** are allowed. In a frame-oriented acquisition, for example, if **startFrame** is 0 and **startScan** is not 0, copying begins at **startScan** within the most recently acquired frame.

timeout is the number of clock ticks (1 tick every 0.055 s) to wait for data that NI-DAQ has not yet acquired. There are two special cases for timeout:

-1:	Wait indefinitely.
0:	Return immediately if NI-DAQ has not acquired the data.

MDAQ_Get

Continued

getBuffer is the memory space you allocate to store the data retrieved from the acquisition buffer. If you are using an AT-DSP2200, **getBuffer** can be a DSP memory handle or float array as well as an integer array or `NI_DAQ_Mem` array. NI-DAQ copies the data from the acquisition buffer to **getBuffer**. If you are sampling more than one input channel, NI-DAQ interleaves the samples from the different channels in **getBuffer** in the following order (here, four channels are being sampled):

(sample 1, channel 0), (sample 1, channel 1), (sample 1, channel 2), (sample 1, channel 3), (sample 2, channel 0), (sample 2, channel 1), (sample 2, channel 2), (sample 2, channel 3), (sample 3, channel 0), (sample 3, channel 1), and so on.

numGotten returns the number of items actually copied from the acquisition buffer to **getBuffer**. When **scansOrFrames** is 1, **numGotten** indicates the number of complete frames that NI-DAQ transferred. When **scansOrFrames** is 0, **numGotten** indicates the number of scans that NI-DAQ transferred. If **numGotten** is 0, then NI-DAQ did not copy any data.

lastFrame returns the number of the last frame from which NI-DAQ copied data. If **lastFrame** is 0, then NI-DAQ did not copy any data.

lastScan returns the last scan number within **lastFrame** that NI-DAQ copied from the acquisition buffer to **getBuffer**. If **lastScan** is 0, NI-DAQ did not copy any data.

acqDone returns an indication of whether the data acquisition has completed.

0:	The data acquisition is not yet complete.
1:	The acquisition is complete.

If you have chosen either continuous acquisition of scans in the `MDAQ_Setup` function or unlimited acquisition of frames after the first trigger in the `MDAQ_Start` function, the status is never one because data acquisition runs indefinitely until stopped by an `MDAQ_Stop` or `MDAQ_Clear` call.

☞ **Note:** ***C Programmers*—numGotten, lastFrame, lastScan, *and* acqDone *are pass-by-reference parameters.***

Using This Function

Call `MDAQ_Get` to retrieve data from the acquisition buffer both while acquisition is in progress and after acquisition is complete. When **getOrTap** is set to *GET*, `MDAQ_Get` allows retrieval of all acquired data with no gaps. Because the acquisition buffer is circular, you must retrieve data fast enough to keep pace with the acquisition or NI-DAQ

MDAQ_Get

Continued

may overwrite data. If NI-DAQ overwrites data before retrieving it, NI-DAQ returns an **overWriteError** warning along with the latest block of data. Sequential data retrieval then resumes from this new position in the acquisition buffer.

If NI-DAQ returns an overwrite error, NI-DAQ overwrote data as the function was copying the data to the **getBuffer** and the data in **getBuffer** may be corrupted.

A *GET* never returns data that NI-DAQ has already returned by a previous *GET*. A *TAP* returns data that NI-DAQ has already returned by a previous *TAP* if the acquisition is slow enough or has stopped.

Unless `MDAQ_Clear` has been called, `MDAQ_Get` always returns pretrigger data, whether complete frames or a partial frame, in correct chronological order, that is, *unwrapped*.

If the acquisition is in pretrigger mode, `MDAQ_Get` does not *GET* data from the frame currently in progress, but only from completed frames. However, `MDAQ_Get` does *TAP* into a frame in progress during a pretrigger acquisition.

Even though a **timeOutError** or an **overWriteError** warning is returned by `MDAQ_Get`, the function may also return some valid data. The **lastFrame**, **lastScan**, and **numGotten** parameters indicate how much data and where in the acquisition buffer the data came from.

MDAQ_ScanRate

Format

status = MDAQ_ScanRate (deviceNumber, scanInterval, timebase)

Purpose

Selects the data acquisition scan rate—the rate at which all selected input channels are sampled.

Parameters

Input

Name	**Type**		**Description**
	Windows	**Windows NT**	
deviceNumber	I16	I32	assigned by configuration utility
scanInterval	U16	U32	amount of time between scans
timebase	I16	I32	resolution for the sample interval

Parameter Discussion

scanInterval indicates the length of the scan interval (that is, the amount of time to elapse between samples on any one channel).

Range:	2 through 65,535 on the EISA-A2000.
	1, 2, 4, 8 on the AT-A2150 and AT-DSP2200.

On the EISA-A2000, the scan interval is a function of the timebase resolution selected by **timebase**. The following formula determines the actual scan interval in seconds:

scanInterval $*$ (timebase resolution)

MDAQ_ScanRate

Continued

On the AT-A2150 and AT-DSP2200, the following formula determines the actual sampling frequency in hertz:

$$\frac{\text{timebase frequency}}{\textbf{scan interval}}$$

timebase is the resolution to use for the sample-interval counter on the EISA-A2000 or the frequency to use on the AT-A2150 and AT-DSP2200. **timebase** has the following possible values:

- For the EISA-A2000:

-1:	200 ns.
0:	Reserved.
1:	1 µs.
2:	10 µs.
3:	100 µs.
4:	1 ms.
5:	10 ms.

- For the AT-A2150C:

0:	51.2 kHz.
1:	48 kHz.
2:	44.1 kHz.
3:	32 kHz.

- For the AT-A2150S:

0:	Reserved.
1:	24 kHz.
2:	20 kHz.
3:	16 kHz.

- For the AT-DSP2200:

0:	51.2 kHz.
1:	48 kHz.
2:	44.1 kHz.
3:	32 kHz.

MDAQ_ScanRate

Continued

Table 2-7 gives the minimum scan rate values on the EISA-A2000.

Table 2-7. Minimum Scan Rate Values on the EISA-A2000

numChans	timebase	scanInterval	Actual Interval
1	-1	5	1 µs
2	-1	10	2 µs
4	-1	20	4 µs

The EISA-A2000 defaults to a timebase of -1 and a **scanInterval** of 20.

The AT-A2150C defaults to a timebase of 3 and a **scanInterval** of 1 (32 kHz).

The AT-A2150S defaults to a timebase of 3 and a **scanInterval** of 1 (16 kHz).

The AT-DSP2200 defaults to a timebase of 3 and a **scanInterval** of 1 (32 kHz).

MDAQ_Setup

Format

status = MDAQ_Setup (deviceNumber, bufferSize, scansOrFrames, preTrigScans, postTrigScans, acqBuffer)

Purpose

Selects how much data to buffer in memory, how much data to acquire for each trigger, and whether the acquisition is scan oriented or frame oriented.

Parameters

Input

Name	Type		Description
	Windows	Windows NT	
deviceNumber	I16	I32	assigned by configuration utility
bufferSize	U32	U32	size of **acqBuffer**
scansOrFrames	I16	I32	whether the acquisition is scan oriented or frame oriented
preTrigScans	U32	U32	number of scans before a trigger
postTrigScans	U32	U32	number of scans after a trigger
acqBuffer	[I16], [F32], HDL	[I16], [F32], HDL	memory space for the acquisition buffer

Parameter Discussion

bufferSize indicates the size of **acqBuffer** (the acquisition buffer) used during the data acquisition. **bufferSize** is either the number of scans or the number of frames in the acquisition buffer depending on the setting of **scansOrFrames**. Calculating the size of the acquisition buffer in bytes is explained in **acqBuffer**.

scansOrFrames indicates whether the acquisition is scan oriented or frame oriented.

MDAQ_Setup

Continued

0: Scan-oriented acquisition; a single posttrigger frame is acquired and **bufferSize** indicates the number of scans in the acquisition buffer.

1: Frame-oriented acquisition; **bufferSize** indicates the number of frames in the acquisition buffer.

preTrigScans is the number of scans before a trigger to collect in each frame. Values of 1 and 2 for **preTrigScans** are invalid and `MDAQ_Setup` returns the **preTrigScansErr** error code.

postTrigScans is the number of scans after a trigger to collect in each frame. Values of 1 and 2 for **postTrigScans** are invalid and `MDAQ_Setup` returns the **postTrigScansErr** error code.

When **scansOrFrames** = 1 (frame-oriented), the frame size is **preTrigScans** + **postTrigScans**. The acquisition buffer contains **bufferSize** number of frames of this size.

Table 2-8 shows the valid combinations of settings for **scansOrFrames**, **preTrigScans**, and **postTrigScans** and their relationship to acquisition modes.

Table 2-8. Valid Combinations of MDAQ_Setup Parameters

Parameter Combination	Results
scansOrFrames = 1 **preTrigScans** = 0 **postTrigScans** > 0	Frame-oriented data acquisition. Posttrigger mode. Frame size = **postTrigScans** ∗ **numChans**. **bufferSize** is in frames.
scansOrFrames = 1 **preTrigScans** > 0 **postTrigScans** > 0	Frame-oriented data acquisition. Pretrigger mode. Frame size = (**postTrigScans** + **preTrigScans**) ∗ **numChans**. **bufferSize** is in frames
scansOrFrames = 0 **preTrigScans** = 0 **postTrigScans** > 0	Scan-oriented data acquisition. Posttrigger mode (only valid mode). **bufferSize** is in scans. **postTrigScans** is total number of scans to acquire. Use **numTriggers** = 1 in `MDAQ_Start` call.

MDAQ_Setup

Continued

Table 2-8. Valid Combinations of MDAQ_Setup Parameters

Parameter Combination	Results
scansOrFrames = 0 **preTrigScans** = 0 **postTrigScans** = 0	Scan-oriented data acquisition. Posttrigger mode (only valid mode). **bufferSize** is in scans; unlimited number of posttrigger scans is acquired. Use **numTriggers** = 1 in `MDAQ_Start` call.
Note: *When* **scansOrFrames** = *0, setting* **preTrigScans** *greater than 0 returns the* **preTrigScansErr** *error. When* **scansOrFrames** = *1, setting* **postTrigScans** *equal to 0 returns the* **postTrigScansErr** *error*.	

acqBuffer is the memory space you allocated for the acquisition buffer. If you are using an AT-DSP2200, **acqBuffer** can be a DSP memory handle or float array as well as an integer array or `NI_DAQ_Mem` array. Notice that in all cases the buffer is assumed to be the size indicated in the **bufferSize** parameter. For example, if you want a buffer to hold 10 frames, where

numChans (from `MAI_Setup`) = 2 (two channels)
preTrigScans = 500
postTrigScans = 2,000

then you want to allocate 100,000 bytes of storage (as explained in the following equation) and set **bufferSize** to 10.

(**preTrigScans** + **postTrigScans**) ∗ (**numChans**) ∗ (number of frames) ∗ (2 bytes per sample)
= (500 + 2,000) ∗ 2 ∗ 10 ∗ 2
= 100,000 bytes

If you are using an AT-DSP2200 and have enabled floating point translation using `DSP2200_Config`, your samples will be four bytes in size instead of two.

Conversely, if you want a buffer to hold 20,000 scans for a four-channel acquisition, be sure to set **bufferSize** to 20,000 and allocate 160,000 bytes of storage, as explained in the following equation:

(number of channels) ∗ (number of scans) ∗ (2 bytes per sample) = 4 ∗ 20,000 ∗ 2 = 160,000 bytes

MDAQ_Setup

Continued

If you are sampling more than one input channel, NI-DAQ interleaves the samples from the different channels in **acqBuffer** in the following order (here, four channels are being sampled):

> (sample 1, channel 0), (sample 1, channel 1), (sample 1, channel 2), (sample 1, channel 3), (sample 2, channel 0), (sample 2, channel 1), (sample 2, channel 2), (sample 2, channel 3), (sample 3, channel 0), (sample 3, channel 1), and so on.

MDAQ_Start

Format

status = MDAQ_Start (deviceNumber, numTriggers)

Purpose

Starts a multiple-channel data acquisition operation.

Parameters

Input

Name	**Type**		**Description**
	Windows	**Windows NT**	
deviceNumber	I16	I32	assigned by configuration utility
numTriggers	U32	U32	number of triggers that are recognized

Parameter Discussion

numTriggers is the number of triggers that NI-DAQ recognizes and for which NI-DAQ acquires data. If you have set **scansOrFrames** to 1 (frames) in `MDAQ_Setup`, **numTriggers** indicates the number of frames to acquire. The acquisition runs until NI-DAQ has acquired **numTriggers** number of frames, at which time the acquisition stops (as if you had called `MDAQ_Stop`). Setting **numTriggers** to 0 means that the acquisition is in *unlimited* frame acquisition mode. Unlimited frame acquisition mode indicates that the driver responds to triggers and acquires frames of data indefinitely. In this case, the acquisition does not end until you call either `MDAQ_Stop` or `MDAQ_Clear`.

If you have set **scansOrFrames** to 0 (scan-oriented data acquisition) in `MDAQ_Setup`, you *must* set **numTriggers** to 1 (NI-DAQ acquires only one posttrigger frame) and NI-DAQ acquires scans after it receives the first trigger. Furthermore, if you have set **postTrigScans** to 0 (**preTrigScans** must be 0 when **scansOrFrames** is 0), the device acquires an unlimited number of scans until you call either `MDAQ_Stop` or `MDAQ_Clear`.

MDAQ_Stop

Format

status = MDAQ_Stop (deviceNumber)

Purpose

Stops the data acquisition but leaves all settings in effect.

Parameters

Input

Name	Type		Description
	Windows	Windows NT	
deviceNumber	I16	I32	assigned by configuration utility

Using This Function

`MDAQ_Stop` stops the acquisition process by configuring the device to ignore any subsequent sample clock pulses regardless of their source. `MDAQ_Stop` does not prepare the system for the termination of your application and maintains the full functionality of the `MDAQ_Get` call (that is, after an `MDAQ_Stop` call, `MDAQ_Get` returns *unwrapped* pretrigger data). `MDAQ_Stop` is useful to simply stop the acquisition in order to change one or more of the acquisition settings and then restart the acquisition. Remember that changing the number of channels scanned affects the size of both the acquisition buffer and the buffer used in the `MDAQ_Get` call.

MDAQ_StrGet

Format

status = MDAQ_StrGet (deviceNumber, scansOrFrames, getOrTap, numToGet, startFrame, startScan, timeout, getBuffer, numGotten, lastFrame, lastScan, acqDone)

Purpose

Transfers acquired data from the acquisition buffer into the buffer you specified while data acquisition is in progress or after data acquisition is complete. `MDAQ_StrGet` can retrieve data from anywhere in the acquisition buffer. This function is the same as `MDAQ_Get` but is intended for use with BASIC applications to retrieve data and save the data on disk through the BASIC `PUT` statement.

Parameters

Input

Name	Type		Description
	Windows	Windows NT	
deviceNumber	I16	I32	assigned by configuration utility
scansOrFrames	I16	I32	retrieve scans or frames
getOrTap	I16	I32	the retrieval method used
numToGet	U32	U32	number of scans or frames to retrieve
startFrame	U32	U32	frame number to begin
startScan	U32	U32	scan number to begin
timeout	I32	I32	number of clock ticks to wait for data

MDAQ_StrGet

Continued

Output

Name	Type		Description
	Windows	Windows NT	
getBuffer	HDL, [U8]	HDL, [U8]	memory space to store the data retrieved
numGotten	U32	U32	the number of items actually copied
lastFrame	U32	U32	returns the number of the last frame
lastScan	U32	U32	returns the last scan number
acqDone	I16	I32	whether the data acquisition has completed

Parameter Discussion

deviceNumber is assigned by configuration utility of the DAQ device.

scansOrFrames indicates whether to retrieve scans or frames from the acquisition **getBuffer**. NI-DAQ can retrieve scans from either a scan-oriented or frame-oriented acquisition (see `MDAQ_Setup`). NI-DAQ can retrieve frames only from a frame-oriented acquisition.

0: Get scans.
1: Get frames.

getOrTap indicates the retrieval method used to retrieve data from the acquisition buffer.

0: (performing a *GET*)—Get oldest data sequentially.
1: (performing a *TAP*)—Get most recently acquired data.

In either case, you can indicate a starting frame and scan. When performing a *GET*, NI-DAQ uses two internal sequential retrieval pointers (one for the starting frame and one for the starting scan) to keep track of what NI-DAQ has retrieved to guarantee sequential access. NI-DAQ maintains no such pointers when performing a *TAP*.

MDAQ_StrGet

Continued

numToGet is the number of scans or frames to retrieve from the acquisition buffer. If **scansOrFrames** is 0 (retrieving scans) and the acquisition is frame oriented, **numToGet** ranges from 1 to the number of scans in a frame (**preTrigScans** + **postTrigScans**). NI-DAQ cannot retrieve scans across frame boundaries. Always check the **numGotten** parameter to determine the actual number of scans NI-DAQ retrieved. If **scansOrFrames** is 1, **numToGet** ranges from 1 to the number of frames in the acquisition buffer. The **numGotten** parameter indicates the number of frames actually retrieved.

startFrame is the frame number to begin copying from. If the acquisition is scan oriented, NI-DAQ ignores **startFrame** so the following cases are only relevant for frame-oriented acquisitions. If performing a *GET* (**getOrTap** = 0) and **startFrame** is not 0, copying begins in **startFrame**. If performing a *GET* and **startFrame** is 0, NI-DAQ uses the retrieval frame pointer to determine where to begin copying. In both cases, after retrieving the data, NI-DAQ resets the retrieval frame pointer to the frame immediately following the last frame copied. If performing a *TAP* (**getOrTap** = 1) and **startFrame** is not 0, copying begins in **startFrame**. If performing a *TAP* and **startFrame** is 0, NI-DAQ copies the most recently acquired data. When performing a *TAP*, NI-DAQ does not use or modify any internal retrieval pointers.

startScan is the scan number to begin copying from within **startFrame**. If **scansOrFrames** is 1 (retrieving frames), NI-DAQ ignores **startScan**. For the following cases, **scansOrFrames** is 0 (retrieving scans). If performing a *GET* and **startScan** is not 0, copying begins at **startScan** (**startFrame** must also be not 0 or NI-DAQ returns an error). If performing a *GET* and **startScan** is 0, copying begins at the scan pointed to by the retrieval scan pointer (**startFrame** must also be 0 or NI-DAQ returns an error). In both cases, after retrieving the data, NI-DAQ resets the retrieval scan pointer to the scan immediately following the last scan copied. If performing a *TAP* and **startScan** is not 0, copying begins at **startScan**. If performing a *TAP* and **startScan** is 0, NI-DAQ copies the most recently acquired data. When performing a *TAP*, any combination of valid values for **startScan** and **startFrame** are allowed. In a frame-oriented acquisition, for example, if **startFrame** is 0 and **startScan** is not 0, copying begins at **startScan** within the most recently acquired frame.

timeout is the number of clock ticks (1 tick every 0.055 s) to wait for data that NI-DAQ has not yet acquired. There are two special cases for timeout:

-1:	Wait indefinitely.
0:	Return immediately if NI-DAQ has not acquired the data.

MDAQ_StrGet

Continued

getBuffer is the memory space you allocated to store the data retrieved from the acquisition buffer. NI-DAQ copies the data from the acquisition buffer to **getBuffer**. If you are sampling more than one input channel, NI-DAQ interleaves the samples from the different channels in **getBuffer** in the following order (here, four channels are being sampled):

(sample 1, channel 0), (sample 1, channel 1), (sample 1, channel 2), (sample 1, channel 3), (sample 2, channel 0), (sample 2, channel 1), (sample 2, channel 2), (sample 2, channel 3), (sample 3, channel 0), (sample 3, channel 1), and so on.

numGotten returns the number of items NI-DAQ actually copied from the acquisition buffer to **getBuffer**. When **scansOrFrames** is 1, **numGotten** indicates the number of complete frames that NI-DAQ transferred. When **scansOrFrames** is 0, **numGotten** indicates the number of scans that NI-DAQ transferred. If **numGotten** is 0, NI-DAQ did not copy any data.

lastFrame returns the number of the last frame from which NI-DAQ copied data. If **lastFrame** is 0, NI-DAQ did not copy any data.

lastScan returns the last scan number within **lastFrame** that NI-DAQ copied from the acquisition buffer to **getBuffer**. If **lastScan** is 0, NI-DAQ did not copy any data.

acqDone returns an indication of whether the data acquisition has completed.

0:	The data acquisition is not yet complete.
1:	The acquisition is complete.

If you have chosen either continuous acquisition of scans in the `MDAQ_Setup` function or unlimited acquisition of frames after the first trigger in the `MDAQ_Start` function, then the status is never 1 because data acquisition runs indefinitely until stopped by an `MDAQ_Stop` or `MDAQ_Clear` call.

☞ **Note:** ***C Programmers*—numGotten, lastFrame, lastScan, *and* acqDone *are pass-by-reference parameters.***

MDAQ_StrGet

Continued

Using This Function

Call `MDAQ_StrGet` to retrieve data from the acquisition buffer both while acquisition is in progress and after acquisition is complete. When **getOrTap** is set to *GET*, `MDAQ_StrGet` allows retrieval of all acquired data with no gaps. Because the acquisition buffer is circular, you must retrieve data fast enough to keep pace with the acquisition or NI-DAQ may overwrite data. If NI-DAQ overwrite data before retrieving it, NI-DAQ returns an **overWriteError** warning along with the latest block of data. Sequential data retrieval then resumes from this new position in the acquisition buffer.

If NI-DAQ returns an **overWriteError**, NI-DAQ overwrote data as the function was copying the data to the **getBuffer**, and the data in **getBuffer** may be corrupted.

A *GET* never returns data that NI-DAQ has already returned by a previous *GET*. A *TAP* returns data that NI-DAQ has already returned by a previous *TAP* if the acquisition is slow enough or has stopped.

Unless you have called `MDAQ_Clear`, `MDAQ_StrGet` always returns pretrigger data, whether complete frames or a partial frame, in correct chronological order, that is, *unwrapped*.

If the acquisition is in pretrigger mode, `MDAQ_StrGet` does not *GET* data from the frame currently in progress, but only from completed frames. However, `MDAQ_StrGet` does *TAP* into a frame in progress during a pretrigger acquisition.

Even if `MDAQ_StrGet` returns a **timeOutError** or an **overWriteError** warning, the function may also return some valid data. The **lastFrame**, **lastScan**, and **numGotten** parameters indicate how much data and where in the acquisition buffer the data came from.

MDAQ_Trig_Delay

Format

status = MDAQ_Trig_Delay (deviceNumber, delayInterval, timebase)

Purpose

Selects the time to delay after a trigger is received before acquiring data (Posttrigger mode only).

Parameters

Input

Name	Type		Description
	Windows	Windows NT	
deviceNumber	I16	I32	assigned by configuration utility
delayInterval	U16	U32	time to wait after the trigger before acquiring data
timebase	I16	I32	resolution to be used for the delay counter

Parameter Discussion

delayInterval is the time to wait after the trigger before acquiring data in posttrigger mode. Setting **delayInterval** to 0 causes NI-DAQ to acquire data immediately after the trigger occurs. On the EISA-A2000, setting **delayInterval** to 65,535 and **timebase** to 5 (10-ms timebase) causes the acquisition to start 655.35 s, or 10.9 min, after the trigger (this is the longest possible delay).

Range:	0, 3 through 65,535 for the EISA-A2000 and AT-A2150.
	0 through 65,535 for the AT-DSP2200.

timebase is the resolution to be used for the delay counter. **timebase** has the following possible values:

- For the EISA-A2000:

-1:	200 ns.
0:	Reserved.

MDAQ_Trig_Delay

Continued

1:	1 µs.
2:	10 µs.
3:	100 µs.
4:	1 ms.
5:	10 ms.

For the AT-A2150 and AT-DSP2200, you must set **timebase** to 0. The **timebase** used for the trigger delay is the same **timebase** set in `MDAQ_ScanRate`. NI-DAQ ignores the **timebase** parameter here.

NI-DAQ ignores **timebase** if **delayInterval** is 0. The default is no posttrigger delay.

MDAQ_Trig_Select

Format

status = MDAQ_Trig_Select (deviceNumber, digitalTrig, edge, analogTrig, slope, atrigLevel, atrigSource)

Purpose

Selects the trigger source and configures the analog and digital trigger conditions.

Parameters

Input

Name	Type		Description
	Windows	Windows NT	
deviceNumber	I16	I32	assigned by configuration utility
digitalTrig	I16	I32	enables or disables the digital trigger input
edge	I16	I32	selects edge of the digital trigger input
analogTrig	I16	I32	enables or disables the analog trigger circuitry
slope	I16	I32	selects slope condition
atrigLevel	I16	I32	input value that generates a trigger
atrigSource	I16	I32	analog trigger source

Parameter Discussion

digitalTrig enables or disables the digital trigger input (DTRIG) on the I/O connector.

0: Digital trigger disabled.
1: Digital trigger enabled.

MDAQ_Trig_Select

Continued

2:	Drive the trigger signal out the I/O connector (A2150 and DSP2200 only).

edge selects which edge of the digital trigger input (DTRIG) signal generates a trigger.

0:	Trigger on falling edge.
1:	Trigger on rising edge. This option is not available on the AT-A2150.

NI-DAQ ignores **edge** unless **digitalTrig** is 1.

analogTrig enables or disables the analog trigger circuitry**.**

0:	Analog trigger disabled.
1:	Analog trigger enabled.

slope selects which slope condition at the selected analog trigger input generates a trigger.

0:	Trigger on negative slope.
1:	Trigger on positive slope.

NI-DAQ ignores **slope** unless **analogTrig** is 1.

atrigLevel is the code for the input value of the selected analog trigger signal that generates a trigger.

Range:	-2,048 to 2,047 on the EISA-A2000 which corresponds to a ±5.12 V analog trigger range in 2.5 mV steps. For example, **atrigLevel** = 800 corresponds to a voltage trigger level of +2 V. -32,768 to 32,767 on the AT-A2150 and AT-DSP2200 which corresponds to a ±2.828 V analog trigger range.

NI-DAQ ignores **atrigLevel** unless **analogTrig** is 1.

atrigSource selects the analog trigger source as follows:

0:	Analog input channel 0.
1:	Analog input channel 1.
2:	Analog input channel 2 (EISA-A2000 and AT-A2150).
3:	Analog input channel 3 (EISA-A2000 and AT-A2150).
4:	External analog trigger input ATRIG (EISA-A2000 only).

NI-DAQ ignores **atrigSource** unless **analogTrig** is 1.

Using This Function

Call `MDAQ_Trig_Select` to configure trigger circuitry for hardware triggered acquisition. For pretrigger data acquisition, you *must* select a hardware trigger. If you

MDAQ_Trig_Select

Continued

enable neither analog nor digital trigger, NI-DAQ starts the acquisition immediately by a software trigger when you call `MDAQ_Start`. Thus, NI-DAQ acquires only posttrigger data. When you enable both analog and digital triggers, NI-DAQ recognizes the first trigger condition met as the trigger and ignores the other trigger condition within each frame. You can also control triggering from the RTSI bus (see the *RTSI Bus Trigger Functions* section of Chapter 3, *Software Overview*, of the *NI-DAQ User Manual for PC Compatibles*). For the AT-A2150 and AT-DSP2200 you can create a trigger hysteresis window by calling `Trigger_Window_Config`.

MIO_Calibrate

Format

status = MIO_Calibrate (deviceNumber, calOP, saveNewCal, EEPROMloc, calRefChan, DAC0chan, DAC1chan, calRefVolts, refLoc)

Purpose

Note: ***If you have an E Series device, use*** `Calibrate_E_Series`**.**

You can use this function to calibrate your AT-MIO-16F-5, AT-MIO-64F-5, and AT-MIO-16X devices. You need to calibrate your device:

- If it is operating in an environment with a temperature that differs by more than 10˚ C from the temperature at which the device was calibrated. Your device is calibrated at the factory at room temperature (25˚ C).
- Once every year.

You can perform a new calibration or use an existing set of calibration constants by copying the constants from their storage location in the onboard EEPROM. You can also store calibration constants. NI-DAQ automatically loads the calibration constants stored in the EEPROM load area the first time you call a function pertaining to the AT-MIO-16F-5, AT-MIO-64F-5, or AT-MIO-16X devices that requires calibration constants to be loaded (when you call an `AI`, `AO`, `DAQ`, `SCAN`, or `WFM` function).

The load area for the AT-MIO-16F-5 is user area 5. The load area for the AT-MIO-64F-5 and AT-MIO-16X is user area 8.

Warning: ***Read the calibration chapter in your device user manual before using*** `MIO_Calibrate`**.**

MIO_Calibrate

Continued

Parameters

Input

Name	Type		Description
	Windows	Windows NT	
deviceNumber	I16	I32	assigned by configuration utility
calOP	I16	I32	operation to be performed
saveNewCal	I16	I32	save new calibration constants
EEPROMloc	I16	I32	storage location
calRefChan	I16	I32	AI channel that the calibration voltage is connected to
DAC0chan	I16	I32	AI channel that DAC0 is connected to
DAC1chan	I16	I32	AI channel that DAC1 is connected to
calRefVolts	F64	F64	DC calibration voltage
refLoc	I16	I32	source of the internal voltage reference constants

Parameter Discussion

calOP determines the operation to be performed.

1: Load calibration constants from **EEPROMloc**.

2: Calibrate the ADC using internal reference voltage calibration constants in **refLoc**.

3: Calibrate the DACs using internal voltage calibration constants in **refLoc**; **DAC0chan** and **DAC1chan** are the analog input channels to which DAC0 and DAC1 are connected, respectively.

MIO_Calibrate

Continued

4: Calibrate the internal reference voltage. You must connect a DC voltage of **calRefVolts** to the analog input channel **calRefChan**. The calibration constants are always stored in **refLoc**.
5: Copy ADC calibration constants from **EEPROMloc** to EEPROM load area.
6: Copy DAC calibration constants from **EEPROMloc** to EEPROM load area.

☞ **Note:** ***(AT-MIO-16F-5 users only) When* calOp *is 3, you must connect each DAC to the negative side of the respective input channel. Otherwise, the calibration will not converge.***

saveNewCal is only valid when **calOP** is 2 or 3.
0: Do not save new calibration constants in **EEPROMloc**.
1: Save new calibration constants in **EEPROMloc**.

EEPROMloc selects the storage location in the onboard EEPROM. You can use different sets of calibration constants to compensate for configuration or environmental changes.

For the AT-MIO-16F-5:
1: User calibration area 1.
2: User calibration area 2.
3: User calibration area 3.
4: User calibration area 4.
5: User calibration area 5 (initial load area).
6: Factory calibration area (you cannot write into this area).

For the AT-MIO-64F-5 and AT-MIO-16X:
1: User calibration area 1.
2: User calibration area 2.
3: User calibration area 3.
4: User calibration area 4.
5: User calibration area 5.
6: User calibration area 6.
7: User calibration area 7.
8: User calibration area 8 (initial load area).
9: Factory calibration area for unipolar (you cannot write to this area).
10: Factory calibration area for bipolar (you cannot write to this area).

MIO_Calibrate

Continued

calRefChan is the analog input channel that the calibration voltage is connected to when **calOP** is 4.
Range: 0 through 7.

DAC0chan is the analog input channel that DAC0 is connected to when **calOP** is 3. This parameter is not applicable to the AT-MIO-64F-5 because its DAC0 is internally wrapped back.
Range: 0 through 7.

DAC1chan is the analog input channel that DAC1 is connected to when **calOP** is 3. This parameter is not applicable to the AT-MIO-64F-5 because its DAC0 is internally wrapped back.
Range: 0 through 7.

calRefVolts is the value of the DC calibration voltage connected to **calRefChan** when **calOP** is 4.
Range: +6 to +10 V.

refLoc is the source of the internal voltage reference constants when **calOp** is 2 or 3. When **calOP** is 4, NI-DAQ stores the internal voltage reference constants in **refLoc**.

1:	User reference area 1.
2:	User reference area 2.
3:	User reference area 3 (AT-MIO-16X and AT-MIO-64F-5 only).
4:	User reference area 4 (AT-MIO-16X and AT-MIO-64F-5 only).
6:	Factory reference area (you cannot write to this area).

Using This Function

Note: ***Calibration of your MIO or AI device takes some time. Do not be alarmed if the*** `MIO_Calibrate` ***function takes several seconds to execute.***

You should set aside at least 18,000 bytes in BASIC programs for this function to perform calibration. This is done with the `SETMEM` statement. For example:

```
Heap.Size = SETMEM(-18000)
```

Unless you have previously stored new internal voltage reference constants in **refLoc** (the user reference area) 1 or 2 by calling `MIO_Calibrate` with **calOp** set to 4, you must use **refLoc** 6 (the factory reference area) when performing an ADC or a DAC (**calOp** set to 2 or 3, respectively) calibration.

MIO_Calibrate

Continued

A calibration performed in bipolar mode is not valid for unipolar and vice versa. `MIO_Calibrate` performs a bipolar or unipolar calibration, or loads the bipolar or unipolar constants, depending on the value of the polarity parameter in the last call to `AI_Configure`. Because you can configure the AT-MIO-16X and AT-MIO-64F-5 polarities on a per-channel basis, `MIO_Calibrate` uses channel 0 to determine the polarity of the ADC calibration. If you take analog input measurements with the wrong set of calibration constants loaded, you may get erroneous data.

When you use an AT-MIO-16F-5 with **calOp** = 3 (calibrate DACs), you must connect the outputs of the DAC in reverse to the A/D inputs (positive to negative and vice versa). If you do not make the connections properly, the calibration will fail to converge.

If you have altered the device input polarity by the `AI_Configure` call, NI-DAQ will automatically reload the correct calibration constants. Refer to the description of `AI_Configure` function for details. Please see the calibration chapter of your device user manual for more information regarding calibrating the device.

☞ **Note:** ***You should always calibrate the ADC and the DACs after calibrating the internal reference voltage.***

MIO_Config

Format

status = MIO_Config (deviceNumber, dither, useAMUX)

Purpose

Turns dithering (the addition of Gaussian noise to the analog input signal) on and off, for an E Series device (except the AT-MIO-16XE-50), AT-MIO-16F-5, AT-MIO-64F-5, and Lab and 1200 series devices (except the Lab-PC+). This function also lets you specify whether to use AMUX-64T channels or onboard channels for the AT-MIO-64F-5 and AT-MIO-64E-3.

Parameters

Input

Name	Type		Description
	Windows	Windows NT	
deviceNumber	I16	I32	assigned by configuration utility
dither	I16	I32	whether to add approximately 0.5 LSB rms of white Gaussian noise to the input signal
useAMUX	I16	I32	whether to use AMUX-64T input channels or AT-MIO-64F-5 onboard channels

Parameter Discussion

dither indicates whether to add approximately 0.5 LSB rms of white Gaussian noise to the input signal. This is useful for applications that involve averaging to increase the effective resolution of a device. For high-speed applications that do not involve averaging, dithering is not recommended and should be disabled.

0: Disable dithering.
1: Enable dithering.

MIO_Config

Continued

This parameter is ignored for the AT-MIO-16XE-50. Dithering is always enabled on this device.

useAMUX is valid for the AT-MIO-64F-5 and AT-MIO-64E-3 only.

1:	If you want to use AMUX-64T channels.
0:	If you want to use onboard channels.

Using This Function

If you want to use the AMUX-64T with the AT-MIO-64F-5 or AT-MIO-64E-3, you must call this function to specify whether to use the AMUX-64T input channels or the AT-MIO-64F-5 or AT-MIO-64E-3 onboard channels. For example, if you have one AMUX-64T device connected to the MIO connector of the AT-MIO-64F-5 or AT-MIO-64E-3, channel numbers 16 through 63 are duplicated. If you want to use AMUX-64T channel 20, you must call `MIO_Config` with **useAMUX** set to 1. Later, if you decide to use onboard channel 20, you must call `MIO_Config` with **useAMUX** set to 0.

NI_DAQ_Mem_Alloc

Format

status = NI_DAQ_Mem_Alloc (handle, elementSize, numElements, memType, memSource)

Purpose

Allocates an array of size (**numElements**)(**elementSize**) bytes and returns a handle to the array. Arrays allocated by `NI_DAQ_Mem_Alloc` are referred to elsewhere in this manual as `NI_DAQ_Mem` arrays.

Parameters

Input

Name	Type		Description
	Windows	Windows NT	
elementSize	I16	I32	the size of each element in the array
numElements	U32	U32	the number of elements in the array
memType	I16	I32	type of memory to allocate
memSource	I16	I32	source of memory

Output

Name	Type		Description
	Windows	Windows NT	
handle	I16	I32	unique handle to the array

NI_DAQ_Mem_Alloc

Continued

Parameter Discussion

elementSize is the size in bytes of each element in the array. **elementSize** must be a multiple of 2 if **memType** is 3. **elementSize** must be a multiple of 4 if **memType** is 4.

numElements is the number of elements in the array.

memType is the type of memory to allocate.

3: 16-bit integer DSP memory (AT-DSP2200 onboard memory).
4: 32-bit floating-point DSP memory (AT-DSP2200 onboard memory).

If **memType** is 4 (floating-point DSP memory), NI-DAQ translates data copied to the buffer from IEEE single-precision floating-point format to 32C floating-point format. NI-DAQ translates data copied from the buffer from 32C floating-point format to IEEE single-precision floating-point format.

memSource is the source of the memory. If **memType** is 3 or 4 (DSP memory), **memSource** is the device number of the DSP board with the DSP memory.

handle is the handle for the allocated array that you need to use when calling other `NI_DAQ_Mem` functions. These functions are `NI_DAQ_Mem_Attributes`,

`NI_DAQ_Mem_Copy`, `NI_DAQ_Mem_Free`, `NI_DAQ_Mem_Lock`, and `NI_DAQ_Mem_Unlock`.

☞ **Note:** ***C Programmers*—handle *is a pass-by-reference parameter.***

Using This Function

`NI_DAQ_Mem_Alloc` allocates memory for an array. The handle returned by `NI_DAQ_Mem` can only be used by other `NI_DAQ_Mem` functions; it cannot be passed to functions that expect an array. To use an array allocated through `NI_DAQ_Mem_Alloc` with a function like `DAQ_Start`, call `NI_DAQ_Mem_Lock` and pass the locked handle returned from that function to `DAQ_Start` as a buffer.

NI_DAQ_Mem_Attributes

Format

status = NI_DAQ_Mem_Attributes (handle, elementSize, numElements, lockCnt, memType, memSource)

Purpose

Returns the attributes of the given memory handle.

Parameters

Input

Name	Type		Description
	Windows	Windows NT	
elementSize	I16	I32	the size of each element in the array

Output

Name	Type		Description
	Windows	Windows NT	
elementSize	I16	I32	the size of each element in the array
numElements	U32	U32	the number of elements in the array
lockCnt	I16	I32	number of open locks on **handle**
memType	I16	I32	type of memory
memSource	I16	I32	source of memory

NI_DAQ_Mem_Attributes

Continued

Parameter Discussion

handle is the handle to an array allocated through `NI_DAQ_Mem_Alloc`.

elementSize is the size in bytes of each element in the array.

numElements is the number of elements in the array.

lockCnt is the number of times the memory handle has been locked.

memType is the type of memory allocated.

memSource is the source of the memory.

☞ **Note:** ***C Programmers*—elementSize, numElements, lockCnt, memType, *and* memSource *are pass-by-reference parameters.***

Using This Function

This procedure simply returns the parameters that were passed to `NI_DAQ_Mem_Alloc`, with the exception of the **lockCnt** parameter, which changes as the array is locked and unlocked.

NI_DAQ_Mem_Copy

Format

status = NI_DAQ_Mem_Copy (handle, buffer, startIndex, numEl, CopyDirection)

Purpose

Copies data from or to memory allocated by `NI_DAQ_Mem_Alloc`.

Parameters

Input

Name	Type		Description
	Windows	Windows NT	
handle	I16	I32	the handle to the allocated array
startIndex	U32	U32	index into the array
numEl	U32	U32	the number of elements to copy
CopyDirection	I16	I32	flag indicating direction of copy

Input/Output

Name	Type		Description
	Windows	Windows NT	
buffer	[I16], HDL	[I16], HDL	buffer to which data is saved or from which data is taken

Parameter Discussion

handle is the handle returned by `NI_DAQ_Mem_Alloc`.

NI_DAQ_Mem_Copy

Continued

startIndex and **numEl** indicate which elements of the array specified by **handle** NI-DAQ will copy. NI-DAQ will copy the elements indexed from **startIndex** to **startIndex** + (**numEl** - 1).

copyDirection indicates whether NI-DAQ will copy data from the array specified by **handle** to **buffer** or from **buffer** to the array.

0:	Data is copied from **handle** to **buffer**.
1:	Data is copied from **buffer** to **handle**.

buffer is where NI-DAQ will copy data from or to. It is your responsibility to ensure that the data type of the buffer and of the data stored in the allocated memory are the same. Because this function can work with any data type, NI-DAQ cannot type check. It is also your responsibility to ensure that the buffer is large enough to perform the copy.

If **buffer** is a DSP handle allocated by NI-DSP, `NI_DAQ_Mem_Copy` assumes that it is a floating-point buffer and translates the data from 32C to IEEE single-precision format.

Using This Function

Use this function to copy data from a user program variable or array to an array created by `NI_DAQ_Mem_Alloc` or vice versa. NI-DAQ can copy data one element at a time or in large blocks. It is more efficient to copy large blocks of data at once instead of doing multiple single element copies. `NI_DAQ_Mem_Copy` can copy any amount of data, assuming that both the buffer and the array are large enough.

`NI_DAQ_Mem_Copy` only supports DSP memory buffers that are allocated using the `NI_DAQ_Mem_Alloc` function. Using a buffer allocated by the `DSP_Alloc` function returns an error code.

NI-DAQ keeps track of whether the buffer is an integer or floating point buffer. This data type information is initially specified by the **memType** parameter in the `NI_DAQ_Mem_Alloc` function, and will be maintained by subsequent NI-DAQ function calls that operate on this buffer. It is your responsibility to ensure that data placed into this buffer is the correct type when using non-NI-DAQ functions. If NI-DAQ places the wrong type of data into the buffer, the translation by the copy operation will be incorrect. This will not return an error code.

Alternatively, you can use DSP buffers allocated with either NI-DAQ or NI-DSP allocation functions with the `DSP_CopyMem` function. You will need to provide the data type information to `DSP_CopyMem` via the type parameter.

NI_DAQ_Mem_Free

Format

status = NI_DAQ_Mem_Free (handle)

Purpose

To free the memory allocated by a call to `NI_DAQ_Mem_Alloc`. You can free only unlocked memory. The number of calls to `NI_DAQ_Mem_Unlock` must match the number of calls to `NI_DAQ_Mem_Lock` before you call `NI_DAQ_Mem_Free`.

Parameters

Input

Name	**Type**		**Description**
	Windows	**Windows NT**	
handle	I16	I32	the handle to the allocated array

Parameter Discussion

handle is the handle returned by `NI_DAQ_Mem_Alloc`.

Using This Function

You should call this function when you no longer need an allocated array. After you call this function, **handle** is no longer valid. Lock handles returned by `NI_DAQ_Mem_Lock` are also invalid and should not be used after the array is free.

Do not attempt to free memory that is locked. If the memory has been locked multiple times, you must unlock it the same number of times. Call `NI_DAQ_Mem_Attributes` to find out how many locks there are on the memory. You must call `NI_DAQ_Mem_Unlock` that many times before you call `NI_DAQ_Mem_Free`.

NI_DAQ_Mem_Lock

Format

status = NI_DAQ_Mem_Lock (handle, lockedHandle)

Purpose

Increments the lock count for the given handle and returns the locked handle of the memory allocated by `NI_DAQ_Mem_Alloc`. You must lock an `NI_DAQ_Mem` array before you can use it.

Parameters

Input

Name	**Type**		**Description**
	Windows	**Windows NT**	
handle	I16	I32	the handle to the allocated array

Output

Name	**Type**		**Description**
	Windows	**Windows NT**	
lockedHandle	HDL	HDL	locked handle to the allocated memory

Parameter Discussion

handle is the handle returned by `NI_DAQ_Mem_Alloc`.

lockedHandle is a locked handle to the allocated memory. NI-DAQ can pass the parameter to other functions that expect an array. In Visual Basic, NI-DAQ must pass the locked handle by value (preceded by the word *ByVal* in the procedure call) to functions such as `DAQ_Op` that expect arrays.

NI_DAQ_Mem_Lock

Continued

Using This Function

Memory allocated by `NI_DAQ_Mem_Alloc` must be locked before it can be accessed by user code or by other NI-DAQ functions. The lock handle returned by `NI_DAQ_Mem_Lock` is valid until you call `NI_DAQ_Mem_Unlock` or `NI_DAQ_Mem_Free`.

NI_DAQ_Mem_Unlock

Format

status = NI_DAQ_Mem_Unlock (handle)

Purpose

Decrements the lock count and unlocks the allocated memory for the given handle.

Parameters

Input

Name	**Type**		**Description**
	Windows	**Windows NT**	
handle	I16	I32	the handle to the allocated array

Parameter Discussion

handle is the handle returned by `NI_DAQ_Mem_Alloc`.

Using This Function

Call this function when you are not accessing the array. This allows better memory management by the operating system. After you call this function, the lock handle returned by `NI_DAQ_Mem_Lock` is no longer valid. You must call `NI_DAQ_Mem_Lock` again to get a valid lock handle.

Do not attempt to unlock memory that was not previously locked.

Peek_DAQ_Event

Format

status = Peek_DAQ_Event (timeOut, handle, message, wParam, lParam)

Purpose

Peeks at the next pending event message defined by `Config_DAQ_Event_Message` in the NI-DAQ message queue.

Parameters

Input

Name	Type		Description
	Windows	Windows NT	
timeOut	U32	I32	number of clock ticks to wait for a message

Output

Name	Type		Description
	Windows	Windows NT	
handle	I16	I32	user-defined handle
message	I16	I32	user-defined message
wParam	I16	I32	device and data acquisition process status
lParam	I32	U32	number of scans done

Parameter Discussion

timeOut is the number of clock ticks (1 tick is about 55 ms) to wait for a message. There is one special case for **timeout**:

0: Check once and return.

Peek_DAQ_Event

Continued

handle is the handle you define that NI-DAQ passes to `Config_DAQ_Event_Message` if **status** = **noError**.

If **status** is 0, the value returned in the **handle** parameter is the same value you passed to `Config_DAQ_Event_Message`, `Config_Alarm_Deadband`, or `Config_ATrig_Event_Message` when you enabled your message. If **status** is not 0, the value returned in the **handle** parameter is 0.

message is the message you define that NI-DAQ passes to `Config_DAQ_Event_Message` if **status** = **noError**. **message** is 0 if an error has occurred.

wParam contains two pieces of information if **status** = **noError**. The lower byte of **wParam** is the device that generated the message and the upper byte of **wParam** is a Boolean flag **doneFlag** indicating whether the data acquisition process has ended.

0:	Data acquisition is still running.
1:	Data acquisition has stopped.

wParam is 0 if an error has occurred.

lParam is the number of the scan when the message is generated if **status** = **noError**. **lParam** is 0 if an error has occurred.

Using This Function

This function looks at the next message from the NI-DAQ message queue without removing it. This is useful when your application is in a lengthy operation. `Peek_DAQ_Event` can periodically peek at the NI-DAQ message queue for pending messages.

This function continuously checks the NI-DAQ message queue until it finds a pending message or a timeout value expires. If **timeOut** is 0, this function checks for an NI-DAQ event message once and returns.

If a message is pending, this function returns the message information.

To remove messages from the queue, call `Get_DAQ_Event`.

RTSI_Clear

Format

status = RTSI_Clear (deviceNumber)

Purpose

Disconnects all RTSI bus trigger lines from signals on the specified device.

Parameter

Input

Name	Type		Description
	Windows	Windows NT	
deviceNumber	I16	I32	assigned by configuration utility

Using This Function

`RTSI_Clear` clears all RTSI bus trigger line connections from the specified device, including a system clock signal connected through a call to `RTSI_Clock` (you can connect or disconnect other device system clocks only by changing jumpers on the devices). After you execute `RTSI_Clear`, the device is neither driving signals onto any trigger line nor receiving signals from any trigger line. You can use this call to reset the device RTSI bus interface.

RTSI_Clock

Format

status = RTSI_Clock (deviceNumber, connect, dir)

Purpose

Connects or disconnects the system clock from the RTSI bus if the device can be programmed to do so. You can connect or disconnect the other device system clock signals to and from the RTSI bus using jumper settings.

Parameters

Input

Name	**Type**		**Description**
	Windows	**Windows NT**	
deviceNumber	I16	I32	assigned by configuration utility
connect	I16	I32	connect or disconnect the system clock
dir	I16	I32	direction of the connection

Parameter Discussion

connect indicates whether to connect or disconnect the system clock from the RTSI bus.

0: Disconnect.
1: Connect.

dir indicates the direction of the connection. If **connect** is 0, then **dir** is meaningless.

0: Receive clock signal from the RTSI bus trigger line.
1: Transmit clock signal to the RTSI bus trigger line.

RTSI_Clock

Continued

Using This Function

`RTSI_Clock` can connect the onboard system clock of an AT-MIO-16X, AT-MIO-64F-5, AT-AO-6/10, AT-A2150, MC-DIO-32F, or EISA-A2000 to the RTSI bus. Calling `RTSI_Clock` with **connect** equal to 1 and **dir** equal to 1 configures the specified **deviceNumber** to transmit its system clock signal onto the RTSI bus. You do not need to specify a RTSI bus trigger line because NI-DAQ uses a dedicated line. Calling `RTSI_Clock` with **connect** equal to 1 and **dir** equal to 0 configures the specified **deviceNumber** to use the signal on the RTSI bus dedicated clock pin as this device system clock. In this way, the two devices are controlled by a single system clock.

Calling `RTSI_Clock` with **connect** equal to 0 disconnects the clock signal from the RTSI bus. `RTSI_Clear` also disconnects the clock signal from the RTSI bus.

`RTSI_Clock` always returns an error if **deviceNumber** is not an AT-MIO-16X, AT-MIO-64F-5, AT-AO-6/10, AT-A2150, MC-DIO-32F, or EISA-A2000. To connect the system clock signal of any other device to the RTSI bus, you must change a jumper setting on the device. See the appropriate user manual for instructions.

☞ **Note:** ***If you are using an E Series device, see the*** `Select_Signal` ***function.***

RTSI_Conn

Format

status = RTSI_Conn (deviceNumber, sigCode, trigLine, dir)

Purpose

Connects a device to the specified RTSI bus trigger line.

Parameters

Input

Name	Type		Description
	Windows	Windows NT	
deviceNumber	I16	I32	assigned by configuration utility
sigCode	I16	I32	signal code number to be connected
trigLine	I16	I32	RTSI bus trigger line
dir	I16	I32	direction of the connection

Parameter Discussion

sigCode is the signal code number of the device signal to be connected to the trigger line. Signal code numbers for each device type are in the *RTSI Bus Trigger Functions* section of Chapter 3, *Software Overview,* of the *NI-DAQ User Manual for PC Compatibles*.

trigLine is the RTSI bus trigger line that is to be connected to the signal.
Range: 0 through 6.

dir is the direction of the connection.

0:	Receive signal (input, receiver) from the RTSI bus trigger line.
1:	Transmit signal (output, source) to the RTSI bus trigger line.

RTSI_Conn

Continued

Using This Function

`RTSI_Conn` programs the RTSI interface on the specified **deviceNumber** such that NI-DAQ connects the signal identified by **sigCode** to the trigger line specified by **trigLine**. For example, if the specified **deviceNumber** is an MIO or AI device, the device **sigCode** is 7, the RTSI **trigLine** is 3, and the **dir** is 1, NI-DAQ drives the output produced by counter 1 (OUT1) on the specified **deviceNumber** onto trigger line 3 of the RTSI bus. You need to make another call to `RTSI_Conn` to program another MIO or AI device (or the same device) to receive the OUT1 signal (**dir** =0) in order to make use of it.

The second call could access another MIO or AI device and use parameters **sigCode** = 0, **trigLine** = 3, and **dir** = 0. This call configures the second MIO or AI device RTSI interface to receive a signal from trigger line 3 and drive it onto the MIO or AI device EXTCONV* signal. The total effect of these two calls is that the MIO or AI device EXTCONV* signal on the second device is controlled by the OUT1 signal on the first MIO or AI device, thus controlling A/D conversions on the second MIO or AI device by a counter on the first.

☞ **Note:** ***If you are using an E Series device, see the*** `Select_Signal` ***function.***

Rules for RTSI Bus Connections

Observe the following rules when routing signals over the RTSI bus trigger lines:

- You can connect any signal to any trigger line.
- RTSI connections should have only one source signal but can have multiple receiver signals. Connecting two or more source signals causes bus contention over the trigger line.
- You can connect two or more signals on the same device together using a RTSI bus trigger line as long as you follow the above rules.

You can disconnect RTSI connections by using either `RTSI_DisConn` or `RTSI_Clear`.

RTSI_DisConn

Format

status = RTSI_DisConn (deviceNumber, sigCode, trigLine)

Purpose

Disconnects a device signal from the specified RTSI bus trigger line.

Parameters

Input

Name	Type		Description
	Windows	Windows NT	
deviceNumber	I16	I32	assigned by configuration utility
sigCode	I16	I32	signal code number
trigLine	I16	I32	RTSI bus trigger line

Parameter Discussion

sigCode is the signal code number of the device signal to be disconnected from the RTSI bus trigger line. Signal code numbers for each device type are in the *RTSI Bus Trigger Functions* section of Chapter 3, *Software Overview*, of the *NI-DAQ User Manual for PC Compatibles*.

trigLine specifies the RTSI bus trigger line that is to be disconnected from the signal.
Range: 0 through 6.

Using This Function

`RTSI_DisConn` programs the RTSI bus interface on the specified **deviceNumber** such that NI-DAQ disconnects the signal identified by **sigCode** and the trigger line specified by **trigLine**.

☞ **Note:** ***It takes the same number of*** `RTSI_DisConn` ***calls to disconnect a connection as it took*** `RTSI_Conn` ***calls to make the connection in the first place. (See*** `RTSI_Conn` ***for further explanation.)***

SC_2040_Config

Format

status = SC_2040_Config (deviceNumber, channel, sc2040gain)

Purpose

Informs NI-DAQ that an SC-2040 Track-and-Hold accessory is attached to the device specified by **deviceNumber** and communicates to NI-DAQ gain settings for one or all channels.

☞ **Note:** ***NI-DAQ for Windows NT does not support this function.***

Parameters

Input

Name	Type		Description
	Windows	Windows NT	
deviceNumber	I16	n/a	assigned by configuration utility
channel	I16	n/a	number of SC-2040 channel you want to configure; use -1 to indicate all SC-2040 channels
sc2040gain	I16	n/a	specifies gain you have set using jumpers on the SC-2040

Parameter Discussion

channel allows you to specify an individual channel on the SC-2040 or all SC-2040 channels.
Range: -1 for all channels and 0 through 7 for individual channels.

sc2040gain allows you to indicate the gain you have selected with your SC-2040 jumpers.
Range: 1, 10, 100, 200, 300, 500, 600, 700, 800.

SC_2040_Config

Continued

Using This Function

You must use this function before any analog input function that uses the SC-2040.

This function reserves PFI 7 line on your E Series device for use by NI-DAQ and the SC-2040. This line is configured for output, and the output is a signal that indicates when a scan is in progress.

Warning: ***Do not attempt to drive the PFI 7 line after calling this function. If you do, you may damage your SC-2040, your E Series device, and your equipment.***

Example 1:

You have set the jumper for a gain of 100 for all your SC-2040 channels. You should call `SC_2040_Config` as follows:

```
SC_2040_Config(deviceNumber, -1, 100)
```

Example 2:

You have set the jumper for a gain of 100 for channels 0, 3, 4, 5, and 6 on your SC-2040, gain 200 for channels 1 and 2, and gain 500 for channel 7. You should call function `SC_2040_Config` several times as follows:

```
SC_2040_Config(deviceNumber, -1, 100)

SC_2040_Config(deviceNumber, 1, 200)

SC_2040_Config(deviceNumber, 2, 200)

SC_2040_Config(deviceNumber, 7, 500)
```

SCAN_Demux

Format

status = SCAN_Demux (buffer, count, numChans, numMuxBrds)

Purpose

Rearranges, or demultiplexes, data acquired by a `SCAN` operation into row-major order (that is, each row of the array holding the data corresponds to a scanned channel) for easier access by C applications. `SCAN_Demux` does not need to be called by BASIC applications to rearrange two-dimensional arrays because these arrays are accessed in column-major order.

Parameters

Input

Name	Type		Description
	Windows	Windows NT	
count	U32	U32	number of samples
numChans	I16	I32	number of channels that were scanned
numMuxBrds	I16	I32	number of AMUX-64T devices used

Input/Output

Name	Type		Description
	Windows	Windows NT	
buffer	[I16]	[I16]	conversion samples returned

SCAN_Demux

Continued

Parameter Discussion

buffer is an integer array of A/D conversion samples returned by a `SCAN` operation.

count is the integer length of **buffer** (that is, the number of samples contained in **buffer**).

numChans is the number of channels that NI-DAQ scanned when the data was created. If you used SCXI to acquire the data, **numChans** should be the total number of channels sampled during one scan. Otherwise, this parameter is the same as the value of **numChans** selected in `SCAN_Setup`, `Lab_ISCAN_Start`, `SCAN_Op`, or `Lab_ISCAN_Op`.

Range:	1 through 16.
	1 through 512 for the E Series devices, AT-MIO-16F-5, AT-MIO-64F-5, and AT-MIO-16X.

numMuxBrds is the number of AMUX-64T devices used during the multiple-channel acquisition. NI-DAQ ignores this parameter for the DAQCard-500/700 and 516, Lab and 1200 series, and LPM devices.

Range:	0, 1, 2, or 4.

Using This Function

If your **buffer** was initially declared as a two-dimensional array, then after `SCAN_Demux` rearranges your data, you can access any point acquired from any channel by specifying the channel in the first dimension and the data point in the second dimension. For example, suppose NI-DAQ scanned channels 3 and 5 and **buffer** is zero based. Then **buffer**[0][9] contains the 10th data point (numbering starts at zero) scanned from channel 3 (the first of the two channels), and **buffer**[1][14] contains the 15th data point acquired from channel 5.

If the number of channels scanned varies each time you run your program, then you probably should be using a one-dimensional array to hold the data. You can index this array in the following manner after `SCAN_Demux` performs its rearrangement to access any point acquired from any channel (again, suppose that channels 3 and 5 were scanned).

count is the total number of data points acquired.

total_chans is the total number of channels scanned (different from **numChans** if **numMuxBrds** is greater than zero).

SCAN_Demux

Continued

points_per_chan is then the number of data points acquired from each channel (that is, **count**/total_chans).

buffer[0 * points_per_chan + 9] contains the 10th data point scanned from channel 3.

buffer[1 * points_per_chan + 14] contains the 15th data point acquired at channel 5.

SCAN_Op

Format

status = SCAN_Op (deviceNumber, numChans, chans, gains, buffer, count, sampleRate, scanRate)

Purpose

Performs a synchronous, multiple-channel scanned data acquisition operation. `SCAN_Op` does not return until NI-DAQ has acquired all the data or an acquisition error has occurred (MIO and AI devices only).

Parameters

Input

Name	Type		Description
	Windows	Windows NT	
deviceNumber	I16	I32	assigned by configuration utility
numChans	I16	I32	number of channels
chans	[I16]	[I32]	list of channels
gains	[I16]	[I32]	list of gain settings
count	U32	U32	number of samples
sampleRate	F64	F64	desired sample rate in pts/s
scanRate	F64	F64	desired scan rate in scans/s

SCAN_Op

Continued

Output

Name	Type		Description
	Windows	Windows NT	
buffer	[I16], HDL	[I16], HDL	contains the acquired data

Parameter Discussion

numChans is the number of channels listed in the scan sequence.

Range: 1 through 16.
1 through 512 for the E Series devices, AT-MIO-16F-5, AT-MIO-64F-5, and AT-MIO-16X.

chans is an integer array or `NI_DAQ_Mem` array of a length not less than **numChans** that contains the channel scan sequence to be used. **chans** can contain any onboard analog input channel number in any order. For onboard analog input channel ranges, see Table B-1 in Appendix B. For example, if **numChans** = 4 and if **chans**[1] = 7, the second channel to be scanned is analog input channel number 7, and NI-DAQ scans four analog input channels.

☞ **Note:** ***The channels contained in the* chans *array refer to the onboard channel numbers.***

If you use one or more external multiplexer devices (AMUX-64Ts) with any MIO or AI device except the MIO-64, the total number of channels scanned equals (four-to-one multiplexer) ∗ (number of onboard channels scanned) ∗ (number of external multiplexer devices), or the total number of channels scanned equals (4) ∗ (**numChans**) ∗ (num_mux_brds). For example, if you use one AMUX-64T and scan eight onboard channels, the total number of channels scanned equals (4) ∗ (8) ∗ (1) = 32.

If you use one or more external multiplexer devices (AMUX-64Ts) with the MIO-64, the total number of channels scanned equals (4) ∗ (**numChans1**) ∗ (num_mux_brds) + **numChans2**, where:

- 4 represents a four-to-one multiplexer.
- **numChans1** is the number of onboard channels (of an MIO or AI connector) scanned.

Range: 0 through 7 differential, 0 through 15 single-ended.

SCAN_Op

Continued

- num_mux_brds is the number of external multiplexer devices.
- **numChans2** is the number of onboard channels (of an analog connector) scanned.

Range: 0 through 23 differential, 0 through 48 single-ended.

If you are using SCXI, you must scan the appropriate analog input channels on the DAQ device that correspond to the SCXI channels you want. You should select the SCXI scan list using `SCXI_SCAN_Setup` before you call this function. Please refer to the *NI-DAQ User Manual for PC Compatibles* for more information on SCXI channel assignments.

gains is an integer array of a length not less than **numChans** that contains the gain setting to be used for each channel in the scan sequence selected in **chans**. NI-DAQ applies the gain value contained in **gains**[*n*] to the channel number contained in **chans**[*n*] when NI-DAQ scans that channel. This gain setting applies only to the DAQ device; if you use SCXI, you must establish any gain you want at the SCXI module either by setting jumpers on the module or by calling `SCXI_Set_Gain`. Refer to Appendix B, *Analog Input Channel and Gain Settings and Voltage Calculation*, for valid gain settings. If you use an invalid gain, NI-DAQ returns an error.

buffer is an integer array or an `NI_DAQ_Mem` array. **buffer** must have a length not less than **count**. When `SCAN_Op` returns with an error code equal to zero, **buffer** contains the acquired data.

count is the number of samples to be acquired (that is, the number of A/D conversions to be performed).

Range: 3 through 2^{32} - 1 (except the E Series).
2 through $2^{24} *$ (total number of channels scanned) or 2^{32}-1, whichever is less (E Series).

sampleRate is the sample rate you want in units of pts/s. This is the rate at which NI-DAQ samples channels within a scan sequence.

Range: Roughly 0.00153 pts/s through 500,000 pts/s. The maximum rate varies according to the type of device you have.

scanRate is the scan rate you want in units of scans per second (scans/s). This is the rate at which NI-DAQ performs scans. NI-DAQ performs a scan each time the function samples all the channels listed in the scan sequence.

Range: 0 and roughly 0.00153 scans/s up to 500,000 scans/s. A value of 0 means that there is no delay between scans and that the effective **scanRate** is **sampleRate** / **numChans**.

SCAN_Op

Continued

When **scanRate** is not 0, **scanRate** must allow at least 2 µs of delay between the last channel of the scan and the first channel of the next scan. This delay must be at least 11 µs on the AT-MIO-16X and 6 µs on the AT-MIO-16F-5 and AT-MIO-64F-5.

Using This Function

`SCAN_Op` initiates a synchronous process of acquiring A/D conversion samples and storing them in a buffer. `SCAN_Op` does not return control to your application until NI-DAQ acquires all the samples you want (or until an acquisition error occurs). When you use posttrigger mode (with pretrigger mode disabled), the process stores **count** A/D conversions in the buffer and ignores any subsequent conversions.

☞ **Note:** ***If you have selected external start triggering of the data acquisition operation, a high-to-low edge at the STARTTRIG* pin on the I/O connector of the MIO-16 and AT-MIO-16D, or the EXTTRIG* pin on the AT-MIO-16F-5, AT-MIO-64F-5, and AT-MIO-16X initiates the data acquisition operation. If you are using an E Series device, you need to apply a trigger that you select through the*** `Select_Signal` ***or*** `DAQ_Config` ***functions to initiate data acquisition. Be aware that if you do not apply the start trigger,*** `SCAN_Op` ***does not return control to your application. Otherwise,*** `SCAN_Op` ***issues a software trigger to initiate the data acquisition operation.***

If you have enabled pretrigger mode, the sample counter does not begin counting acquisitions until you apply a signal at the stop trigger input. Until you apply this signal, the acquisition remains in a cyclical mode, continually overwriting old data in the buffer with new data. Again, if you do not apply the stop trigger, `SCAN_Op` does not return control to your application.

In any case, you can use `Timeout_Config` to establish a maximum length of time for `SCAN_Op` to execute.

SCAN_Sequence_Demux

Format

status = SCAN_Sequence_Demux (numChans, chanVector, bufferSize, buffer, samplesPerSequence, scanSequenceVector, samplesPerChannelVector)

Purpose

Rearranges the data produced by a multi-rate acquisition so that all the data from each channel is stored in adjacent elements of your buffer.

☞ **Note:** ***NI-DAQ for Windows NT does not support this function.***

Parameters

Input

Name	Type		Description
	Windows	Windows NT	
numChans	I16	n/a	the number of channels
chanVector	[I16]	n/a	the channel list
bufferSize	U32	n/a	the number of samples the buffer holds
samplesPerSequence	I16	n/a	the number of samples in a scan sequence
scanSequenceVector	[I16]	n/a	contains the scan sequence

SCAN_Sequence_Demux

Continued

Input/Output

Name	Type		Description
	Windows	Windows NT	
buffer	[I16]	n/a	the acquired samples

Output

Name	Type		Description
	Windows	Windows NT	
samplesPerChannelVector	[U32]	n/a	the number of samples for each channel

Parameter Discussion

numChans is the number of entries in the **chanVector** and **samplesPerChannelVector** arrays.

chanVector contains the channels sampled in the acquisition that produced the data contained in **buffer**. It may be identical to the channel vector you used in the call to `SCAN_Sequence_Setup` or it may contain the channels in a different order. `SCAN_Sequence_Demux` will reorder the data in **buffer** such that the data for **chanVector**[0] occurs first, the data for **chanVector**[1] occurs second, and so on.

bufferSize is the number of samples in the **buffer**.

buffer is the array containing the data produced by the multi-rate acquisition. When `SCAN_Sequence_Demux` returns, the data in **buffer** will be rearranged.

samplesPerSequence is the number of samples in a scan sequence (obtained from a previous call to `SCAN_Sequence_Setup)` and the size of the **scanSequenceVector** array.

SCAN_Sequence_Demux

Continued

scanSequenceVector contains the scan sequence created by NI-DAQ as a result of a previous call to `SCAN_Sequence_Setup`. You obtain a copy of **scanSequenceVector** by calling `SCAN_Sequence_Retrieve`.

samplesPerChannelVector contains the number of samples for each channel. The channel listed in entry *i* of **chanVector** will have a number of samples equal to the value of **samplesPerChannelVector**[*i*].

Using This Function

`SCAN_Sequence_Demux` rearranges multirate data so that retrieving the data of a channel is more straightforward. The following example illustrates how to use this function:

The input parameters are as follows:
numChans = 3
chanVector = {2, 5, 7}
bufferSize = 14
buffer = {2, 5, 7, 2, 2, 5, 2, 2, 5, 7, 2, 2, 5, 2} where a 2 represents a sample from channel 2, and so on.
samplesPerSequence = 7
scanSequenceVector = {2, 5, 7, 2, 2, 5, 2}

The output parameters are as follows:
buffer = {2, 2, 2, 2, 2, 2, 2, 2, 5, 5, 5, 5, 7, 7} where a 2 represents a sample from channel 2, and so on.
samplesPerChannelVector = {8, 4, 2}

The data from a channel may be located in the buffer by calculating the index of the first sample and the index of the last sample. The data from a channel listed in **chanVect**[0] (channel 2) begins at index 0 and ends at index **samplesPerChannelVector** [0] - 1 (index 7). The first sample for the channel listed in **chanVector**[1] (channel5) begins at **samplesPerChannelVector** [0] (index 8) and ends at (**samplesPerChannelVector** [0] + **samplesPerChannelVector** [1]) - 1 (index 11). The first sample for the channel listed in **chanVector**[2] (channel 7) begins at (**samplesPerChannelVector** [0] + **samplesPerChannelVector** [1]) (index 12) and ends at (**samplesPerChannelVector** [0] + **samplesPerChannelVector** [1] + **samplesPerChannelVector** [2]) - 1 (index 13).

SCAN_Sequence_Retrieve

Format

status = SCAN_Sequence_Retrieve (device, samplesPerSequence, scanSequenceVector)

Purpose

Returns the scan sequence created by NI-DAQ as a result of a previous call to `SCAN_Sequence_Setup`.

☞ **Note:** ***NI-DAQ for Windows NT does not support this function.***

Parameters

Input

Name	**Type**		**Description**
	Windows	**Windows NT**	
device	I16	n/a	assigned by configuration utility
samplesPerSequence	I16	n/a	the number of samples in a scan sequence

Output

Name	**Type**		**Description**
	Windows	**Windows NT**	
scanSequenceVector	[I16]	n/a	contains the scan sequence

Parameter Discussion

samplesPerSequence is the number of samples in a scan sequence (obtained from a previous call to `SCAN_Sequence_Setup)` and the size of the **scanSequenceVector** output parameter.

SCAN_Sequence_Retrieve

Continued

scanSequenceVector contains the scan sequence created by NI-DAQ as a result of a previous call to `SCAN_Sequence_Setup`. The scan sequence will not contain the ghost channel place holders.

Using This Function

`SCAN_Sequence_Retrieve` is used to obtain the actual scan sequence to program the device. You will need this information if you want to call `SCAN_Sequence_Demux` to rearrange your data or if you want to extract particular channels data from your acquisition buffer without rearranging it. If you use `DAQ_Monitor` to extract the data of a channel, you do not need the actual scan sequence.

SCAN_Sequence_Setup

Format

status = SCAN_Sequence_Setup (device, numChans, chanVector, gainVector, scanRateDivisorVector, scansPerSequence, samplesPerSequence)

Purpose

Initializes the device for a multirate scanned data acquisition operation. Initialization includes selecting the channels to be scanned, assigning gains to these channels and assigning different sampling rates to each channel by dividing down the base scan rate.

☞ **Note:** ***NI-DAQ for Windows NT does not support this function.***

Parameters

Input

Name	Type		Description
	Windows	Windows NT	
device	I16	n/a	assigned by configuration utility
numChans	I16	n/a	number of channels
chanVector	[I16]	n/a	channel scan sequence
gainVector	[I16]	n/a	gain setting to be used for each channel in **chanVector**

SCAN_Sequence_Setup

Continued

Output

Name	Type		Description
	Windows	Windows NT	
scansPerSequence	I16	n/a	the number of scans in a scan sequence
samplesPerSequence	I16	n/a	the number of samples in a scan sequence

Parameter Discussion

numChans is the number of entries in the three input vectors. All three input vectors must have the same number of entries.

chanVector contains the onboard channels that will be scanned. A channel cannot be listed more the once. Refer to Appendix B, *Analog Input Channel and Gain Settings and Voltage Calculation*, for valid channel settings.

gainVector contains the gain settings to be used for each channel in **chanVector**. The channel listed in entry *i* of **chanVector** will have the gain listed in entry *i* of **gainVector.**

scanRateDivisorVector contains the scan rate divisors to be used for each channel. The sample rate for a channel equals the base scan rate (that is the scan rate specified when `SCAN_Start` is called) divided by the scan rate divisor for that channel. The channel listed in entry *i* of **chanVector** will have the scan rate divisor listed in entry *i* of **scanRateDivisorVector.**

scansPerSequence is an output parameter that contains the total number of scans in the scan sequence created by NI-DAQ from your **chanVector** and **scanRateDivisorVector** including any scans that consist entirely of *ghost channels*, or place holders.

samplesPerSequence is an output parameter that contains the total number of samples in the scan sequence excluding any *ghost channel* place holders. The total size of a scan sequence including ghost channel place holders is limited by the size of the memory on your device used to hold this information. Currently this limit is 512 entries. Because **samplesPerSequence** excludes ghost channel place holders, it is possible to get an error even if **samplesPerSequence** is less than 512.

SCAN_Sequence_Setup

Continued

Using This Function

You must observe the following restrictions:

- Interval scanning must be used.
- A channel can be listed only once in the channel vector.
- SCXI cannot be used.
- The AMUX-64T device cannot be used.
- Your acquisition cannot be pretriggered.
- The size of your buffer (the value of the count parameter to `SCAN_Start`) must be a multiple of **samplesPerSequence.**

The following example shows how to use `SCAN_Sequence_Setup`:

numChans = 3
chanVector = {2, 5, 7}
gainVector = {1, 1, 1}
scanRateDivisorVector = {1, 2, 4}

The scan rate divisor for channel 2 is 1 so it will be sampled at the base scan rate. The scan rate divisor for channel 5 is 2 so it will be sampled at a rate equal to the base scan rate divided by 2. Likewise, the scan rate divisor for channel 7 is 4 so it will be sampled at a rate equal to the base scan rate divided by 4.

The scan sequence created by NI-DAQ looks like this:

scan number:	1	2	3	4
channels sampled:	2, 5, 7	2	2, 5	2

scansPerSequence = 4
samplesPerSequence = 7

If your base scan rate is 1,000 scans/s, channel 2 is sampled at 1,000 S/s, channel 5 is sampled at 500 S/s, and channel 7 is sampled at 250 S/s.

ScansPerSequence and **samplesPerSequence** are used to calculate the size of your acquisition buffer. Your buffer size must be an integer multiple of **samplesPerSequence**. Use **ScansPerSequence** if you want to size your buffer to hold some unit of time's worth of data. For example, to figure out the size of a buffer in units of samples and to hold *N* seconds of data, use the following formula:

bufferSize = *N* * (*scanRate* / **scansPerSequence**) * **samplesPerSequence**

SCAN_Sequence_Setup

Continued

The *bufferSize* returned by the above formula will have to be rounded up so that it is a multiple of the **samplesPerSequence** if **scansPerSequence** does not divide evenly into *scanRate*.

In this example, your buffer size must be a multiple of 7. The number of samples your buffer must hold to contain 5 s of data at a base scan rate of 1,000 scans/s is:

5 * (1,000 / 4) * 7 = 8,750 S.

SCAN_Setup

Format

status = SCAN_Setup (deviceNumber, numChans, chanVector, gainVector)

Purpose

Initializes circuitry for a scanned data acquisition operation. Initialization includes storing a table of the channel sequence and gain setting for each channel to be digitized (MIO and AI devices only).

Parameters

Input

Name	Type		Description
	Windows	Windows NT	
deviceNumber	I16	I32	assigned by configuration utility
numChans	I16	I32	number of channels
chanVector	[I16]	[I32]	channel scan sequence
gainVector	[I16]	[I32]	gain setting to be used for each channel in **chanVector**

Parameter Discussion

numChans is the number of channels in the **chanVector**.

Range:	1 through 16.
	1 through 512 for the E Series devices, AT-MIO-16F-5, AT-MIO-64F-5, and AT-MIO-16X.

chanVector is an integer array of length **numChans** that contains the onboard channel scan sequence to be used. **chanVector** can contain any analog input channel number in any order. For the channel number range refer to Table B-1 in Appendix B. For example, if **numChans** = 4 and if **chanVector**[1] = 7, then the second channel to be scanned is analog input channel 7, and four analog input channels are scanned.

SCAN_Setup

Continued

☞ **Note:** ***The channels listed in the scan sequence refer to the onboard channel numbers.***

If you use one or more external multiplexer devices (AMUX-64Ts) with any MIO or AI device except the MIO-64, the total number of channels scanned equals (four-to-one multiplexer) ∗ (number of onboard channels scanned) ∗ (number of external multiplexer devices), or the total number of channels scanned equals (4) ∗ (**numChans**) ∗ (num_mux_brds). For example, if you use one AMUX-64T and scan eight onboard channels, the total number of channels scanned equals (4) ∗ (8) ∗ (1) = 32.

If you use one or more external multiplexer devices (AMUX-64Ts) with the MIO-64, the total number of channels scanned equals (4) ∗ (**numChans**1) ∗ (num_mux_brds) + **numChans**2, where:

- 4 represents four-to-one multiplexer.
- **numChans**1 is the number of onboard channels (of an MIO or AI connector) scanned.
  Range: 0 through 7 differential, 0 through 15 single-ended.
- num_mux_brds is the number of external multiplexer devices.
- **numChans**2 is the number of onboard channels (of an analog connector) scanned.
  Range: 0 through 23 differential, 0 through 48 single-ended.

If you are using SCXI, you must scan the analog input channels on the DAQ device that corresponds to the SCXI channels you want. You should select the SCXI scan list using `SCXI_SCAN_Setup` before you call this function. Refer to the *NI-DAQ User Manual for PC Compatibles* for more information on SCXI channel assignments.

gainVector is an integer array of length **numChans** that contains the gain setting to be used for each channel specified in **chanVector**. This gain setting applies only to the DAQ device; if you use SCXI, you must establish any gain you want at the SCXI module either by setting jumpers on the module or by calling `SCXI_Set_Gain`. Refer to Appendix B, *Analog Input Channel and Gain Settings and Voltage Calculation*, for valid gain settings.

For example, if **gainVector**[5] = 10 (assuming an MIO-16L device) then, when NI-DAQ scans the sixth channel, the function sets the gain circuitry to a gain of 10. Notice also that **gainVector**[*i*] corresponds to **chanVector**[*i*]. If **gainVector**[2] = 100 and **chanVector**[2] = 3, then the third channel NI-DAQ scans is analog input channel 3, and the function sets its gain to 100.

SCAN_Setup

Continued

Using This Function

`SCAN_Setup` stores **numChans**, **chanVector**, and **gainVector** in the Mux-Gain Memory table on the device. The function uses this memory table during scanning operations (`SCAN_Start)` to automatically sequence through an arbitrary set of analog input channels and to allow gains to automatically change during scanning.

You need to call `SCAN_Setup` to set up a scan sequence for scanned operations; afterwards, you only need to call the function when you want a scan sequence. If you call `DAQ_Start` or `AI_Read`, NI-DAQ modifies the Mux-Gain Memory table on the device; therefore, you should use `SCAN_Setup` again after NI-DAQ modifies these calls to reinitialize the scan sequence.

SCAN_Start

Format

status = SCAN_Start (deviceNumber, buffer, count, sampTimebase, sampInterval, scanTimebase, scanInterval)

Purpose

Initiates a multiple-channel scanned data acquisition operation, with or without interval scanning, and stores its input in an array (MIO and AI devices only).

Parameters

Input

Name	Type		Description
	Windows	Windows NT	
deviceNumber	I16	I32	assigned by configuration utility
count	U32	U32	number of samples
sampTimebase	I16	I32	resolution used for the sample-interval counter
sampInterval	U16	U32	length of the sample interval
scanTimebase	I16	I32	resolution for the scan-interval counter
scanInterval	U16	U32	length of the scan interval

SCAN_Start

Continued

Output

Name	Type		Description
	Windows	Windows NT	
deviceNumber	I16	I32	assigned by configuration utility

Parameter Discussion

buffer is an integer array or an `NI_DAQ_Mem` array. **buffer** must have a length equal to or greater than **count**.

count is the number of samples to be acquired (that is, the number of A/D conversions to be performed). For double-buffered acquisitions, **count** specifies the size of the buffer, and **count** must be an even number.

Range:	3 through 2^{32} - 1 (except the E Series).
	2 through 2^{24} ∗ (total number of channels scanned) or 2^{32}-1, whichever is less (E Series).

count must be an integer multiple of the total number of channels scanned. **count** refers to the *total* number of A/D conversions to be performed; therefore, the number of samples acquired from each channel is equal to **count** divided by the total number of channels scanned. This is also the total number of scans. For the E Series devices the total number of scans must be at least 2. If you do not use external multiplexer (AMUX-64T) devices, the total number of channels scanned is equal to the value of **numChans**.

If you use one or more external multiplexer devices with any MIO or AI device except the MIO-64, the total number of channels scanned equals (four-to-one multiplexer) ∗ (number of onboard channels scanned) ∗ (number of external multiplexer devices), or the total number of channels scanned equals (4) ∗ (**numChans**) ∗ (num_mux_brds). For example, if you use one AMUX-64T and scan eight onboard channels, the total number of channels scanned equals (4) ∗ (8) ∗ (1) = 32.

SCAN_Start

Continued

If you use one or more external multiplexer devices (AMUX-64Ts) with the MIO-64, the total number of channels scanned equals (4) ∗ (**numChans**1) ∗ (num_mux_brds) + **numChans**2, where:

- 4 represents a four-to-one multiplexer.
- **numChans**1 is the number of onboard channels (of an MIO or AI connector) scanned.
  Range: 0 through 7 differential, 0 through 15 single-ended.
- num_mux_brds is the number of external multiplexer devices.
- **numChans**2 is the number of onboard channels (of an analog connector) scanned.
  Range: 0 through 23 differential, 0 through 48 single-ended.

If you use SCXI, the total number of channels scanned is the total number of channels specified in the `SCXI_SCAN_Setup` call.

sampTimebase selects the clock frequency that indicates the timebase, or resolution, to be used for the sample-interval counter. The sample-interval counter controls the time that elapses between acquisition of samples within a scan sequence.

sampTimebase has the following possible values:

-3:	20 MHz clock used as a timebase (50 ns resolution) (E Series only).
-1:	5 MHz clock used as timebase (200 ns resolution) (AT-MIO-16F-5, AT-MIO-64F-5, and AT-MIO-16X only).
0:	External clock used as timebase (Connect your own timebase frequency to the internal scan-interval counter via the SOURCE5 input for the MIO devices or, by default, the PFI8 input for the E Series devices).
1:	1 MHz clock used as timebase (1 µs resolution).
2:	100 kHz clock used as timebase (10 µs resolution).
3:	10 kHz clock used as timebase (100 µs resolution).
4:	1 kHz clock used as timebase (1 ms resolution).
5:	100 Hz clock used as timebase (10 ms resolution).

On E Series devices, if you use this function with **sampleTimebase** set to 0 must call the `Select_Signal` function with **signal** set to `ND_IN_CHANNEL_CLOCK_TIMEBASE` and **source** set to a value other than `ND_INTERNAL_20_MHZ` and `ND_INTERNAL_100_KHZ` before calling `SCAN_Start` with **sampleTimebase** set to 0; otherwise, `SCAN_Start` will select low-to-high transitions on the PFI8 I/O connector pin as your external sample timebase.

SCAN_Start

Continued

If sample-interval timing is to be externally controlled (**extConv** = 1 or 3, see `DAQ_Config`), NI-DAQ ignores the **sampTimebase** parameter, which can be any value.

sampInterval indicates the length of the sample interval (that is, the amount of time to elapse between each A/D conversion within a scan sequence).
Range: 2 through 65,535.

The sample interval is a function of the timebase resolution. The actual sample interval in seconds is determined by the following formula:

sampInterval ∗ (sample timebase resolution)

where the sample timebase resolution is equal to one of the values of **sampTimebase** as specified above. For example, if **sampInterval** = 25 and **sampTimebase** = 2, the actual sample interval is 25 ∗ 10 µs = 250 µs. The time to complete one scan sequence in seconds is (the actual sample interval) ∗ (number of channels scanned). If the sample interval is to be externally controlled by conversion pulses applied to the EXTCONV* input, the **sampInterval** parameter is ignored and can be any value.

scanTimebase selects the clock frequency that indicates the timebase, or resolution, to be used for the scan-interval counter. The scan-interval counter controls the time that elapses between scan sequences. **scanTimebase** has the following possible values:

-3:	20 MHz clock used as a timebase (50 ns resolution) (E Series only).
-1:	5 MHz clock used as timebase (200 ns resolution) (AT-MIO-16F-5, AT-MIO-64F-5, and AT-MIO-16X only).
0:	External clock used as timebase (Connect your own timebase frequency to the internal scan-interval counter via the SOURCE5 input for the MIO devices or, by default, the PFI8 input for the E Series devices).
1:	1 MHz clock used as timebase (1 µs resolution).
2:	100 kHz clock used as timebase (10 µs resolution).
3:	10 kHz clock used as timebase (100 µs resolution).
4:	1 kHz clock used as timebase (1 ms resolution).
5:	100 Hz clock used as timebase (10 ms resolution).

On E Series devices, if you use this function with **scanTimebase** set to 0, you must call the function `Select_Signal` with **signal** set to `ND_IN_SCAN_CLOCK_TIMEBASE` and **source** set to a value other than `ND_INTERNAL_20_MHZ` and `ND_INTERNAL_100_KHZ` before calling `SCAN_Start` with **scanTimebase** set to 0; otherwise, `SCAN_Start` will select low-to-high transitions on the PFI8 I/O connector pin as your external scan timebase.

SCAN_Start

Continued

scanInterval indicates the length of the scan interval (that is, the amount of time that elapses between the initiation of each scan sequence). NI-DAQ scans all channels in the scan sequence at the beginning of each scan interval.
Range: 0 or 2 through 65,535.

If **scanInterval** equals zero, the time that elapses between A/D conversions and the time that elapses between scan sequences are both equal to the sample interval. That is, as soon as the scan sequence has completed, NI-DAQ restarts one sample interval later. Another advantage of setting **scanInterval** to 0 is that this frees the scan-interval counter (counter 2) for other operations such as waveform generation or general-purpose counting.

The scan interval is a function of the scan timebase resolution. The actual scan interval in seconds is determined by the following formula:

scanInterval ∗ (scan timebase resolution)

where the scan timebase resolution is equal to one of the values of **scanTimebase** as indicated above. For example, if **scanInterval** = 100 and **scanTimebase** = 2, the scan interval is 100 ∗ 10 µs = 1 ms. This number must be greater than or equal to the sum of the total sample interval + 2 µs for most devices. The scan interval for the AT-MIO-16X must be at least 11 µs longer than the total sample interval. The scan interval for the AT-MIO-16F-5 and AT-MIO-64F-5 must be at least 6 µs longer than the total sample interval. If the scan interval is to be controlled by pulses applied to the OUT2 signal, NI-DAQ ignores this parameter (**extConv** = 2 or 3, see `DAQ_Config`).

☞ **Note:** ***The E Series, AT-MIO-16F-5, AT-MIO-64F-5, and AT-MIO-16X support external control of the sample interval even when you use interval scanning. For the MIO-16 and AT-MIO-16D, if the sample interval is to be controlled externally by pulses applied to the EXTCONV* input, you cannot control the scan interval externally. In this case, NI-DAQ scans the channels repeatedly as fast as you apply the external conversion pulses.***

Using This Function

`SCAN_Start` initializes the Mux-Gain Memory table to point to the start of the scan sequence as specified by `SCAN_Setup`. If you did not specify external sample-interval timing by the `DAQ_Config` call, NI-DAQ sets the sample-interval counter to the specified **sampInterval** and **sampTimebase**, sets the scan-interval counter to the specified **scanInterval** and **scanTimebase**, and sets up the sample counter to count the

SCAN_Start

Continued

number of samples acquired and to stop the data acquisition process when the number of samples acquired equals **count**. If you have specified external sample-interval timing, the data acquisition circuitry relies on pulses received on the EXTCONV* input to initiate individual A/D conversions. In this case, NI-DAQ scans the channels repeatedly as fast as you apply the external conversion pulses.

`SCAN_Start` initializes a background data acquisition process to handle storing of A/D conversion samples into the buffer as NI-DAQ acquires them. When you use posttrigger mode (with pretrigger mode disabled), the process stores up to **count** A/D conversion samples into the buffer and ignores any subsequent conversions. NI-DAQ stores the acquired samples into the buffer with the channel scan sequence data interleaved; that is, the first sample is the conversion from the first channel, the second sample is the conversion from the second channel, and so on.

You cannot make the second call to `SCAN_Start` without terminating this background data acquisition process. If a call to `DAQ_Check` returns **daqStopped** = 1, the samples are available and NI-DAQ terminates the process. In addition, a call to `DAQ_Clear` terminates the background data acquisition process. Notice that if a call to `DAQ_Check` returns an error code of -75 or -76, or **daqStopped** = 1, the process is automatically terminated and there is no need to call `DAQ_Clear`.

If you enable pretrigger mode, `SCAN_Start` initiates a cyclical acquisition that continually fills the buffer with data, wrapping around to the start of the buffer once NI-DAQ has written to the entire buffer. When you apply the signal at the stop trigger input, `SCAN_Start` acquires an additional number of samples specified by the **ptsAfterStoptrig** parameter in `DAQ_StopTrigger_Config` and then terminates. Be aware that a scan sequence always completes. Therefore, NI-DAQ always obtains the most recent data point from the final channel in the scan sequence. When you enable pretrigger mode, the length of the buffer, which is greater than or equal to **count**, should be an integral multiple of **numChans**. If you observed this rule, a sample from the first channel in the scan sequence always resides at **index** = 0 in the buffer.

If you have selected external start triggering of the data acquisition operation, a high-to-low edge at the STARTTRIG* I/O connector input on the MIO-16 and AT-MIO-16D, or the EXTTRIG* connector on the AT-MIO-16F-5, AT-MIO-64F-5, and AT-MIO-16X initiates the data acquisition operation after the `SCAN_Start` call is complete. Otherwise, `SCAN_Start` issues a software trigger to initiate the data acquisition operation before returning.

SCAN_Start

Continued

☞ **Note:** ***If your application calls*** `DAQ_Start` ***or*** `SCAN_Start`***, always ensure that you call*** `DAQ_Clear` ***before your application terminates and returns control to the operating system. Unless you make this call (either directly, or indirectly through*** `DAQ_Check` ***or*** `DAQ_DB_Transfer`***), unpredictable behavior can result.***

You must use the `SCAN_Setup` and `SCAN_Start` functions as a pair. Making a single call to `SCAN_Setup` with multiple calls to `SCAN_Start` will fail and return error **noSetupError**.

If you have an SC-2040 connected to your DAQ device, NI-DAQ will ignore the **sampTimebase** and **sampInterval** parameters. NI-DAQ automatically supplies these parameters to optimally match your hardware.

If you select **sampTimebase** = 0 and **scanTimebase** = 0, you must use the same source for both. This requirement is enforced on most MIO devices through hardware because you connect both timebases to the SOURCE5 I/O connector pin. On E Series devices, if you use the `Select_Signal` function to specify the source of an external sample and external scan timebase, you must specify the same source for both timebases.

SCAN_to_Disk

Format

status = SCAN_to_Disk (deviceNumber, numChans, chans, gains, filename, count, sampleRate, scanRate, concat)

Purpose

Performs a synchronous, multiple-channel scanned data acquisition operation and simultaneously saves the acquired data in a disk file. `SCAN_to_Disk` does not return until all the data has been acquired and saved or an acquisition error has occurred (MIO and AI devices only).

Parameters

Input

Name	Type		Description
	Windows	Windows NT	
deviceNumber	I16	I32	assigned by configuration utility
numChans	I16	I32	number of channels
chans	[I16]	[I32]	list of channels
gains	[I16]	[I32]	list of gain settings
filename	STR	STR	name of the data file
count	U32	U32	number of samples
sampleRate	F64	F64	desired sample rate in pts/s
scanRate	F64	F64	desired scan rate in scans/s
concat	I16	I32	enables concatenation of existing file

SCAN_to_Disk

Continued

Parameter Discussion

numChans is the number of channels listed in **chansArray**.

Range: 1 through 16.
1 through 512 for the E Series devices, AT-MIO-16F-5, AT-MIO-64F-5, and AT-MIO-16X.

chans is an integer array of a length not less than **numChans** that contains the onboard channel scan sequence to be used. **chans** can contain any analog input channel number in any order. For channel number ranges, refer to Table B-1 in Appendix B, *Analog Input Channel and Gain Settings and Voltage Calculation*. For example, if **numChans** = 4 and if **chans**[1] = 7, the second channel to be scanned is analog input channel 7, and NI-DAQ scans four analog input channels.

☞ **Note:** ***The channels contained in the* chans *array refer to the onboard channel numbers.***

If you use one or more external multiplexer devices (AMUX-64Ts) with any MIO or AI device except the MIO-64, the total number of channels scanned equals (four-to-one multiplexer) ∗ (number of onboard channels scanned) ∗ (number of external multiplexer devices), or the total number of channels scanned equals (4) ∗ (**numChans**) ∗ (*num_mux_brds*). For example, if you use one AMUX-64T and scan eight onboard channels, the total number of channels scanned equals (4) ∗ (8) ∗ (1) = 32.

If you use one or more external multiplexer devices (AMUX-64Ts) with the MIO-64, the total number of channels scanned equals (4) ∗ (**numChans**1) ∗ (*num_mux_brds*) + **numChans**2, where:

- 4 represents a four-to-one multiplexer.
- **numChans**1 is the number of onboard channels (of an MIO or AI connector) scanned.
  Range: 0 through 7 differential, 0 through 15 single-ended.
- *num_mux_brds* is the number of external multiplexer devices.
- **numChans**2 is the number of onboard channels (of an analog connector) scanned.
  Range: 0 through 23 differential, 0 through 48 single-ended.

SCAN_to_Disk

Continued

If you use SCXI, you must scan the analog input channels on the DAQ device that corresponds to the SCXI channels you want. You should select the SCXI scan list using `SCXI_SCAN_Setup` before you call this function. Refer to the *NI-DAQ User Manual for PC Compatibles* for more information on SCXI channel assignments.

gains is an integer array of a length not less than **numChans** that contains the gain setting to be used for each channel in the scan sequence selected in **chans**. NI-DAQ applies the gain value contained in **gains**[*n*] to the channel number contained in **chans**[*n*] when the function scans that channel. This gain setting applies only to the DAQ device; if you use SCXI, you must establish any gain you want at the SCXI module either by setting jumpers on the module or by calling `SCXI_Set_Gain`. Refer to Appendix B, *Analog Input Channel and Gain Settings and Voltage Calculation*, for valid gain settings. If you use an invalid gain, NI-DAQ returns an error.

count is the number of samples to be acquired (that is, the number of A/D conversions to be performed). The length of your data file should be exactly twice the value of **count**. If you have previously enabled pretrigger mode (by a call to `DAQ_StopTrigger_Config`), NI-DAQ ignores the **count** parameter.

Range:	3 through 2^{32} - 1 (except the E Series).
	2 through 2^{24} (E Series).

sampleRate is the sample rate you want in units of pts/s. This is the rate at which channels are sampled within a scan sequence.

Range:	Roughly 0.00153 pts/s through 500,000 pts/s.

scanRate is the scan rate you want in units of scans/s. This is the rate at which NI-DAQ performs scans. NI-DAQ performs a scan each time the function samples all the channels listed in the scan sequence.

Range:	0 and roughly 0.00153 scans/s through 500,000 scans/s. A value of zero means that there is no delay between scans and that the effective **scanRate** is **sampleRate** / **numChans**.

concat enables concatenation of data to an existing file. Regardless of the value of **concat**, if the file does not exist, NI-DAQ creates the file.

0:	Overwrite file if it exists.
1:	Concatenate new data to an existing file.

SCAN_to_Disk

Continued

Using This Function

SCAN_to_Disk initiates a synchronous process of acquiring A/D conversion samples and storing them in a disk file. The maximum rate varies according to the type of device you have and the speed and degree of fragmentation of your disk storage device. SCAN_to_Disk does not return control to your application until NI-DAQ acquires and saves all the samples you want (or until an acquisition error occurs). When you use posttrigger mode (with pretrigger mode disabled), the process stores **count** A/D conversions in the file and ignores any subsequent conversions.

Note: ***If you have selected external start triggering of the data acquisition operation, a high-to-low edge at the STARTTRIG* I/O connector of the MIO-16 and AT-MIO-16D, or the EXTTRIG* connector of the AT-MIO-16F-5, AT-MIO-64F-5, and AT-MIO-16X initiates the data acquisition operation. If you are using all E Series devices, see the*** `Select_Signal` ***function for information about the external timing signals. Be aware that if you do not apply the start trigger,*** `SCAN_to_Disk` ***does not return control to your application. Otherwise,*** `SCAN_to_Disk` ***issues a software trigger to initiate the data acquisition operation.***

If you have enabled pretrigger mode, the sample counter does not begin counting acquisitions until you apply a signal at the stop trigger input. Until you apply this signal, the acquisition continues to write data into the disk file. NI-DAQ ignores the value of the **count** parameter when you enable pretrigger mode. If you do not apply the stop trigger, SCAN_to_Disk eventually returns control to your application because, sooner or later, you run out of disk space.

In any case, you can use Timeout_Config to establish a maximum length of time for SCAN_to_Disk to execute.

SCXI_AO_Write

Format

status = SCXI_AO_Write (SCXIchassisID, moduleSlot, channel, opCode, rangeCode, voltCurrentData, binaryData, binaryWritten)

Purpose

Sets the DAC channel on the SCXI-1124 module to the specified voltage or current output value. You can also use this function to write a binary value directly to the DAC channel, or to translate a voltage or current value to the corresponding binary value.

☞ **Note:** ***NI-DAQ for Windows NT does not support this function.***

Parameters

Input

Name	Type		Description
	Windows	Windows NT	
SCXIchassisID	I16	n/a	chassis ID number
moduleSlot	I16	n/a	module slot number
channel	I16	n/a	the DAC channel of the module to write to
opCode	I16	n/a	type of data
rangeCode	I16	n/a	the voltage/current range to be used
voltCurrentData	F64	n/a	voltage or current to be produced at the channel
binaryData	I16	n/a	binary value to be written to the DAC

SCXI_AO_Write

Continued

Output

Name	Type		Description
	Windows	Windows NT	
binaryWritten	I16	n/a	actual binary value written to the DAC

Parameter Discussion

channel is the number of the analog output channels on the module.
Range: 0 to 5.

opCode specifies the type of data to write to the DAC channel. You can also use **opCode** to tell `SCXI_AO_Write` to translate a voltage or current value and return the corresponding binary pattern in **binaryWritten** without writing anything to the module.

0: Write a voltage or current to **channel**.
1: Write a binary value directly to **channel**.
2: Translate a voltage or current value to binary, return in **binaryWritten**.

rangeCode is the voltage or current range to be used for the analog output channel.

0: 0 to 1 V.
1: 0 to 5 V.
2: 0 to 10 V.
3: -1 to 1 V.
4: -5 to 5 V.
5: -10 to 10 V.
6: 0 to 20 mA.

voltCurrentData is the voltage or current you want to produce at the DAC channel output. If **opCode** = 1, NI-DAQ ignores this parameter. If **opCode** = 2, this is the voltage or current value you want to translate to binary. If the value is out of range for the given **rangeCode**, `SCXI_AO_Write` returns an error.

binaryData is the binary value you want to write directly to the DAC. If **opCode** is not 1, NI-DAQ ignores this parameter.
Range: 0 to 4,095

SCXI_AO_Write

Continued

binaryWritten returns the actual binary value that NI-DAQ wrote to the DAC. `SCXI_AO_Write` uses a formula given later in this section using calibration constants that are stored on the module EEPROM to calculate the appropriate binary value that will produce the given voltage or current. If **opCode** = 1, **binaryWritten** is equal to **binaryData**. If **opCode** = 2, `SCXI_AO_Write` calculates the binary value but does not write anything to the module.

Using This Function

`SCXI_AO_Write` uses the following equation to translate voltage or current values to binary:

$$B_w = B_l + (V_w - V_l) * (B_h - B_l) / (V_h - V_l)$$

where

B_l = binary value that produces the low value of the range
B_h = binary value that produces the high value of the range
V_h = high value of the range
V_l = low value of the range
V_w = desired voltage or current
B_w = the binary value which will generate V_w

NI-DAQ loads a table of calibration constants from the SCXI-1124 EEPROM load area. The calibration table contains values for B_l and B_h for each channel and range.

The SCXI-1124 is shipped with a set of factory calibration constants in the factory EEPROM area, and a copy of the factory constants in the EEPROM load area. You can recalibrate your module and store your own calibration constants in the EEPROM load area using the `SCXI_Cal_Constants` function. Please refer to the `SCXI_Cal_Constants` function description for calibration procedures and information about the module EEPROM.

If you want to write a binary value directly to the output channel, use **opCode** = 1. `SCXI_AO_Write` will not use the calibration constants or the conversion formula; it will simply write your **binaryData** value to the DAC.

SCXI_Cal_Constants

Format

status = SCXI_Cal_Constants (SCXIchassisID, moduleSlot, channel, opCode, calibrationArea, rangeCode, SCXIgain, DAQboard, DAQchan, DAQgain, TBgain, volt1, binary1, volt2, binary2, calConst1, calConst2)

Purpose

Calculates calibration constants for the given channel and range or gain using measured voltage/binary pairs. You can use this with any SCXI analog input or analog output module. The constants can be stored and retrieved from NI-DAQ memory or the module EEPROM (if your module has an EEPROM). The driver uses the calibration constants to more accurately scale analog input data when you use the `SCXI_Scale` function and output data when you use `SCXI_AO_Write`.

☞ **Note:** ***NI-DAQ for Windows NT does not support this function.***

Parameters

Input

Name	Type		Description
	Windows	Windows NT	
SCXIchassisID	I16	n/a	SCXI chassis ID number
moduleSlot	I16	n/a	SCXI module slot number
channel	I16	n/a	analog input or output channel number
opCode	I16	n/a	operation to perform with the calibration constants
calibrationArea	I16	n/a	where to store or retrieve constants
rangeCode	I16	n/a	the voltage/current range for the analog output channel

SCXI_Cal_Constants

Continued

Name	Type		Description
	Windows	Windows NT	
SCXIgain	F64	n/a	gain setting for the SCXI analog input channel
DAQboard	I16	n/a	device number of DAQ device used to acquire **binary1** and **binary2**
DAQchan	I16	n/a	DAQ device channel number used when acquiring **binary1** and **binary2**
DAQgain	I16	n/a	DAQ device gain code used when acquiring **binary1** and **binary2**
TBgain	F64	n/a	SCXI terminal block gain, if any
volt1	F64	n/a	voltage/current corresponding to **binary1**
binary1	F64	n/a	binary value corresponding to **volt1**
volt2	F64	n/a	voltage/current corresponding to **binary2**
binary2	F64	n/a	binary value corresponding to **volt2**

SCXI_Cal_Constants

Continued

Output

Name	Type		Description
	Windows	Windows NT	
calConst1	F64	n/a	return calibration constant
calConst2	F64	n/a	return calibration constant

Parameter Discussion

channel is the number of the channel on the module.

Range:	0 to *n*-1, where *n* is the number of channels available on the module.
-1:	All channels on the module. For instance, the SCXI-1100 and SCXI-1122 modules have one amplifier for all channels, so calibration constants for those modules apply to all the module channels.
-2:	The voltage (**calConst2**) and current excitation channels (**calConst1**) on the module. This is valid for the SCXI-1122 only, and only when **opCode** = 0.

opCode specifies the type of calibration operation to be performed.

0:	Retrieve calibration constants for the given channel and range or gain from **calibrationArea** and return them in **calConst1** and **calConst2**.
1:	Perform a one-point offset calibration calculation using (**volt1**, **binary1**) for the given channel and gain and write calibration constants to **calibrationArea** (SCXI analog input modules only).
2:	Perform a two-point calibration calculation using (**volt1**, **binary1**) and (**volt2**, **binary2**) for the given channel and range or gain and write calibration constants to **calibrationArea**.
3:	Write the calibration constants passed in **calConst1** and **calConst2** to **calibrationArea** for the given channel and range or gain.
4:	Copy the entire calibration table in **calibrationArea** to the module EEPROM default load area so that it will be loaded automatically into NI-DAQ memory during subsequent application runs (SCXI-1122, SCXI-1124, and SCXI-1141 only).
5:	Copy the entire calibration table in **calibrationArea** to driver memory so NI-DAQ can use the table in subsequent scaling operations in the current NI-DAQ session (SCXI-1122, SCXI-1124, and SCXI-1141 only).

SCXI_Cal_Constants

Continued

calibrationArea is the location NI-DAQ uses for the calibration constants. Read the following *Using This Function* section for an explanation of the calibration table stored in NI-DAQ memory and the SCXI-1122, SCXI-1124, and SCXI-1141 EEPROM organization.

0:	NI-DAQ memory. NI-DAQ maintains a calibration table in memory for use in scaling operations for the module.
1:	Default EEPROM load area. NI-DAQ also updates the calibration table in memory when you write to the default load area (SCXI-1122, SCXI-1124, and SCXI-1141 only)
2:	Factory EEPROM area. You cannot write to this area, but you can read or copy from it (SCXI-1122, SCXI-1124, and SCXI-1141 only).
3:	User EEPROM area (SCXI-1122, SCXI-1124, and SCXI-1141 only).

rangeCode is the voltage or current range of the analog output channel. NI-DAQ only uses this parameter for SCXI analog output modules.

0:	0 to 1 V.
1:	0 to 5 V.
2:	0 to 10 V.
3:	-1 to 1 V.
4:	-5 to 5 V.
5:	-10 to 10 V.
6:	0 to 20 mA.

SCXIgain is the SCXI module or channel gain setting. NI-DAQ only uses this parameter for analog input modules. Valid **SCXIgain** values depend on the module type:

SCXI-1100:	1, 2, 5, 10, 20, 50, 100, 200, 500, 1,000, 2,000.
SCXI-1120:	1, 2, 5, 10, 20, 50, 100, 200, 250, 500, 1,000, 2,000.
SCXI-1121:	1, 2, 5, 10, 20, 50, 100, 200, 250, 500, 1,000, 2,000.
SCXI-1122:	0.01, 0.02, 0.05, 0.1, 0.2, 0.5, 1, 2, 5, 10, 20, 50, 100, 200, 500, 1,000, 2,000.
SCXI-1140:	1, 10, 100, 200, 500
SCXI-1141:	1, 2, 5, 10, 20, 50, 100

DAQboard is the DAQ device you are using with this SCXI module. This applies only when **opCode** = 0, 1, 2, or 3 and **moduleSlot** is an analog input module. Otherwise, set to 0.

DAQchan is the analog input channel of **DAQboard** that you are using with this SCXI module. If you have only one chassis connected to **DAQboard** and **moduleSlot** is in multiplexed mode, **DAQchan** should be 0. **calConst1** will be scaled by the current input

SCXI_Cal_Constants

Continued

range and polarity settings for this channel. This applies only when **opCode** = 0, 1, 2, or 3 and **moduleSlot** is an analog input module. Otherwise, set to 0.

DAQgain is the gain setting for **DAQchan**. It is used to scale **calConst1** (binary offset). This applies only when **opCode** = 0, 1, 2, or 3 and **moduleSlot** is an analog input module. Otherwise, set to 0.

TBgain is the terminal block gain applied to the SCXI channel, if any. Currently, the SCXI-1327 terminal block is the only terminal block that applies gain to your SCXI channels. The SCXI-1327 has switches that you use to select either a gain of 1.0 or a gain of 0.01. You can use this terminal block with an SCXI-1120 or SCXI-1121 module. For terminal blocks that do not apply gain to your SCXI channels, set **TBgain** =1.0.

volt1, **binary1** is the measured voltage/binary pair you have taken for the given channel and range or gain. If the module is analog output, **volt1** is the voltage or current you measured at the output channel after writing the binary value **binary1** to the output channel.

If the module is analog input, **binary1** is the binary value you read from the input channel with a known voltage of **volt1** applied at the input. The **binary1** parameter is floating point, so you may take multiple binary readings from **volt1** and average them to be more accurate and reduce the effects of noise.

volt2, **binary2** is a second measured voltage/binary pair you have taken for the given channel and range or gain. If the module is analog output, **volt2** is the voltage or current you measured when NI-DAQ wrote the binary value **binary2** to the output channel. If the module is analog input, **binary2** is the binary reading from the input channel with a known voltage of **volt2** applied at the input.

calConst1 is the first calibration constant. For analog output modules, **calConst1** is the binary value that will generate the voltage or current at the lower end of the voltage or current range. For analog input modules, **calConst1** is the binary zero offset; that is, the binary reading that would result from an input voltage of zero. The offset is stored as a voltage and must be scaled to a binary value. It is scaled based on **DAQgain** and the current configuration of **DAQchan** (polarity and input range). If **opCode** = 1 or 2, **calConst1** is a return value calculated from the voltage/binary pairs. If **opCode** = 0, **calConst1** is a return constant retrieved from the **calibrationArea**. If **opCode** = 0 and **channel** =-2, **calConst1** is the actual voltage excitation value returned in units of volts. If **opCode** = 3, you should pass your first calibration constant in **calConst1** for NI-DAQ to store in **calibrationArea**.

SCXI_Cal_Constants

Continued

calConst2 is the second calibration constant. For analog output modules, **calConst2** is the binary value that generates the voltage or current at the upper end of the voltage or current range. For analog input modules, **calConst2** is the gain adjust factor; that is, the ratio of the real gain to the ideal gain setting. If **opCode** = 1 or 2, **calConst2** is a return value calculated from the voltage/binary pairs. If **opCode** = 0, **calConst2** is a return constant retrieved from the **calibrationArea**. If **opCode** = 0 and **channel** =-2, **calConst2** is the actual current excitation value returned in units of milliamperes. If **opCode** = 3, you should pass your second calibration constant in **calConst2** for NI-DAQ to store in **calibrationArea**.

Note: ***C Programmers*—calConst1 *and* calConst2 *are pass-by-reference parameters.***

Using This Function

Analog Input Calibration

When you call `SCXI_Scale` to scale binary analog input data, NI-DAQ uses the binary offset and gain adjust calibration constants loaded for the given module, channel, and gain setting to scale the data to voltage. Please refer to the `SCXI_Scale` function description for the equations used.

By default, NI-DAQ loads calibration constants for the SCXI-1122 and SCXI-1141 from the module EEPROM (see the *EEPROM Organization* section later in this function for more information). The SCXI-1141 has only gain adjust constants in the EEPROM and does not have binary zero offset in the EEPROM. All other analog input modules have no calibration constants by default; NI-DAQ assumes no binary offset and ideal gain settings for those modules *unless* you use the following procedure to store calibration constants for your module.

You can determine calibration constants based specifically on your application setup, which includes your type of DAQ device, your DAQ device settings, and your cable assembly, all combined with your SCXI module and its configuration settings.

Note: ***NI-DAQ stores constants in a table for each SCXI module gain setting. If your module has independent gains on each channel, NI-DAQ stores constants for each channel at each gain setting. When you use the following procedure, you are also calibrating for your DAQ device settings, so you must use the same DAQ device settings whenever you use the new calibration constants. The SCXI-1122 and SCXI-1141 factory***

SCXI_Cal_Constants

Continued

EEPROM constants apply only to the SCXI-1122 and SCXI-1141 amplifiers, respectively, so you can use those with any DAQ device setup.

To perform a two-point analog input calibration, perform the following steps:

1. If you are using an AT-MIO-16F-5, AT-MIO-64F-5, or AT-MIO-16X device, you should calibrate your ADC first using the `MIO_Calibrate` function.
2. Make sure the SCXI gain is set to the gain you will be using in your application. If you are using an SCXI-1100, SCXI-1122, or SCXI-1141, you can use the `SCXI_Set_Gain` function, because those modules have software-programmable gain. For other analog input modules, you need to set gain jumpers or DIP switches appropriately.
3. Use `SCXI_Single_Chan_Setup` to program the module for a single-channel operation (as opposed to a channel scanning operation).
4. Ground your SCXI input channel. If you are using an SCXI-1100, SCXI-1122, or SCXI-1141, you can use the `SCXI_Calibrate_Setup` function to internally ground the module amplifier inputs. For other analog input modules, you need to wire the positive and negative channel inputs together at the terminal block.
5. Take several readings using the `DAQ` functions and average them for greater accuracy. You should use the DAQ device gain settings you will be using in your application. If you are using an AT-MIO-16F-5, AT-MIO-64F-5, or AT-MIO-16X, you can enable dither using the `MIO_Config` function to make your averaging more accurate. You should average over an integral number of 60 Hz or 50 Hz power line cycles to eliminate line noise.

   You now have your first volt/binary pair: **volt1** = 0.0, and **binary1** is your binary reading or binary average.
6. Now apply a known, stable, non-zero voltage to your input channel at the terminal block. Preferably, your input voltage should be close to the upper limit of your input voltage range for the given gain setting.
7. Take another binary reading or average. If your binary reading is the maximum binary reading for your DAQ device, you should try a smaller input voltage. This is your second volt/binary pair: **volt2** and **binary2**.
8. Call `SCXI_Cal_Constants` with your two volt/binary pairs and **opCode** = 2. Make sure you pass the correct **SCXIgain** you used and pass the gain code you used in `AI_Read` or `DAQ_Op` in the **DAQgain** parameter.

SCXI_Cal_Constants

Continued

If you are using an SCXI-1122 or SCXI-1141, you can save the constants in the module EEPROM (**calibrationArea** = 1 or 3). Refer to the *EEPROM Organization* section later in this function for information about constants in the EEPROM. It is best to use **calibrationArea** = 3 (user EEPROM area) as you are calibrating, and then call `SCXI_Cal_Constants` again at the end of your calibration sequence with **opCode** = 4 to copy your EEPROM area to the default EEPROM load area. That way there will be two copies of your new constants, and you can revert to the factory constants using **opCode** = 4 without wiping out your new constants entirely.

For other analog input modules, you must specify **calibrationArea** = 0 (NI-DAQ memory). Unfortunately, calibration constants stored in NI-DAQ memory will be lost at the end of the current NI-DAQ session. You may want to create a file and save the constants returned in **calConst1** and **calConst2** so that you can load them again in subsequent application runs using `SCXI_Cal_Constants` with **opCode** = 3.

Any subsequent calls to `SCXI_Scale` for the given module, channel, and gain setting will use the new calibration constants when scaling. You can repeat steps 2 through 8 for any other channel or gain settings you want to calibrate.

You may use a different voltage for the first measurement instead of grounding the input channel. For instance, if you know you will be using a specific input voltage range, you might use the endpoints of your expected input voltage range as **volt1** and **volt2**. Then you would be specifically calibrating your expected input voltage range.

If you are using an SCXI-1100, SCXI-1122, or SCXI-1141, you can perform a one-point calibration to determine the binary offset; you can do this easily without external hookups using the `SCXI_Calibrate_Setup` function to internally ground the amplifier. Use the procedure above, skipping steps 6 and 7, and using **opCode** = 1 for the `SCXI_Cal_Constants` function.

If you are storing calibration constants in the SCXI-1122 or SCXI-1141 EEPROM, your binary offset and gain adjust factors must not exceed the ranges given in the respective module user manuals. The constant format in the EEPROM does not allow for larger constants. If your constants exceed these specifications, the function returns **badExtRefError**. If this error occurs, you should make sure your **SCXIgain**, **DAQgain**, and **TBgain** values are the actual settings you used to measure the volt/binary pairs, and you may want to recalibrate your DAQ device, if applicable.

SCXI_Cal_Constants

Continued

Analog Output Calibration

When you call `SCXI_AO_Write` to output a voltage or current to your SCXI-1124 module, NI-DAQ uses the calibration constants loaded for the given module, channel, and output range to scale the voltage or current value to the appropriate binary value to write to the output channel. By default, NI-DAQ will load calibration constants into memory for the SCXI-1124 from the module EEPROM load area (see the *EEPROM Organization* section for more information).

You can recalibrate your SCXI-1124 module to create your own calibration constants using the following procedure:

1. Use the `SCXI_AO_Write` function with **opCode** = 1. If you are calibrating a voltage output range, pass the parameter **binaryData** = 0. If you are calibrating the 0 to 20 mA current output range (**rangeCode** = 6), pass the parameter **binaryData** = 255.
2. Measure the output voltage or current at the output channel with a voltmeter. This is your first volt/binary pair: **binary1** = 0 or 255 and **volt1** is the voltage or current you measured at the output.
3. Use the `SCXI_AO_Write` function with **opCode** = 1 to write the **binaryData** = 4,095 to the output DAC.
4. Measure the output voltage or current at the output channel. This is your second volt/binary pair: **binary2** = 4,095 and **volt2** is the voltage or current you measured at the output.
5. Call `SCXI_Cal_Constants` with your voltage/binary pairs and **opCode** = 2. You can save the constants on the module EEPROM (**calibrationArea** = 1 or 3). Refer to the following *EEPROM Organization* section for information about constants in the EEPROM. It is best to use **calibrationArea** = 3 (user EEPROM area) as you are calibrating, and then call `SCXI_Cal_Constants` again at the end of your calibration sequence with **opCode** = 4 to copy the user EEPROM area to the default load area. That way there will be two copies of your new constants and you can revert to the factory constants using **opCode** = 4 without wiping out your new constants entirely.

Repeat the procedure above for each channel and range you want to calibrate. Subsequent calls to `SCXI_AO_Write` will use your new constants to scale voltage or current to the correct binary value.

SCXI_Cal_Constants

Continued

EEPROM Organization

The SCXI-1122, SCXI-1124, and SCXI-1141 modules have an onboard EEPROM to handle storage of calibration constants. The EEPROM is divided into three areas:

- The **factory area** is shipped with a set of factory calibration constants; you cannot write into the factory area, but you can read from it.
- The **default load area** is where NI-DAQ automatically looks to load calibration constants the first time you access the module during an NI-DAQ session using an NI-DAQ function call, such as `SCXI_Reset`, `SCXI_Single_Chan_Setup`, or `SCXI_AO_Write`. When the module is shipped, the default load area contains a copy of the factory calibration constants. When you write to the default load area using `SCXI_Cal_Constants`, NI-DAQ also updates the constants in NI-DAQ memory.
- The **user area** is an area for you to store your own calibration constants that you calculate by following the instructions above and using the `SCXI_Cal_Constants` function. You can also put a copy of your own constants in the default load area if you want NI-DAQ to automatically load your constants for subsequent NI-DAQ sessions.

SCXI_Calibrate_Setup

Format

status = SCXI_Calibrate_Setup (SCXIchassisID, moduleSlot, calOp)

Purpose

Used to ground the amplifier inputs of an SCXI-1100, SCXI-1122, or SCXI-1141 so that you can determine the amplifier offset. You can also use this function to switch a shunt resistor across your bridge circuit to test the circuit. Shunt calibration is supported for the SCXI-1122 or SCXI-1121 modules with the SCXI-1321 terminal block.

Parameters

Input

Name	**Type**		**Description**
	Windows	**Windows NT**	
SCXIchassisID	I16	I32	logical ID assigned to the SCXI chassis
moduleSlot	I16	I32	chassis slot number
calOp	I16	I32	calibration mode

Parameter Discussion

calOp indicates the calibration mode you want.

0: Disable calibration.

1: Connect the positive and negative inputs of the SCXI-1100, SCXI-1122, or SCXI-1141 amplifier(s) together and to analog reference.

2: Switch the shunt resistors across the bridge circuit on the SCXI-1121 (Revision C or later) or SCXI-1122.

SCXI_Calibrate_Setup

Continued

Using This Function

The zero offset of the SCXI-1100, SCXI-1122, or SCXI-1141 amplifiers varies with the module gain. Once you know the offset at a specific gain setting, you can add that offset to any readings acquired at that gain. In general, the procedure for determining the offset at a particular gain is as follows:

1. `SCXI_Single_Chan_Setup`—Enable the module output, route the module output on the SCXIbus if necessary, and resolve any SCXIbus contention if necessary. The module channel you specify is irrelevant.
2. `SCXI_Set_Gain`—Set the module gain to the setting that you will use in your application.
3. `SCXI_Calibrate_Setup`—Ground the amplifier inputs.
4. Acquire data using the `DAQ` functions; you should acquire and average many samples. If you have enabled the filter on the module, wait for the amplifier to settle after calling `SCXI_Calibrate_Setup` before you acquire data. Refer to your SCXI-1100, SCXI-1122, or SCXI-1141 user manuals for settling times caused by filter settings.
5. `SCXI_Calibrate_Setup`—Disable calibration.
6. Continue with your application. Whenever you acquire samples from the module at the gain that you chose in step 2, subtract the binary offset that you read in step 4 from each sample before scaling the data, or call `SCXI_Cal_Constants` to store the offset in NI-DAQ memory or the EEPROM. Then subsequent calls to `SCXI_Scale` for the given gain will automatically subtract the offset for you. Refer to the `SCXI_Cal_Constants` function for more information.

Refer to your SCXI-1321 or SCXI-1122 user manuals for information about how the module applies the shunt resistor(s) when **calOp** = 2.

The SCXI-1141 has a separate amplifier for each channel, so you will have to repeat the above procedure for each channel you wish to calibrate.

SCXI_Change_Chan

Format

status = SCXI_Change_Chan (SCXIchassisID, moduleSlot, moduleChan)

Purpose

Selects a new channel of a multiplexed module that you have previously set up for a single-channel analog input operation using the `SCXI_Single_Chan_Setup` function.

Parameters

Input

Name	**Type**		**Description**
	Windows	**Windows NT**	
SCXIchassisID	I16	I32	logical ID assigned to the SCXI chassis
moduleSlot	I16	I32	chassis slot number of the module
moduleChan	I16	I32	channel number

Parameter Discussion

moduleChan is the channel number of the new input channel on the module that is to be read.

Range:	0 to *n*-1, where *n* is the number of input channels on the module.
-1:	Set up to read the temperature sensor on the terminal block connected to the module if the temperature sensor is in the MTEMP configuration.

Using This Function

It is important to realize that this function affects only the channel selection on the module. It does not affect the module output enable or any analog signal routing on the SCXIbus; the `SCXI_Single_Chan_Setup` function is required to do that. `SCXI_Change_Chan` can be very useful in applications like those shown in Chapter 3, *Software Overview*, of the *NI-DAQ User Manual for PC Compatibles*, especially when you are trying to read several channels on a module in a loop at relatively high speeds.

SCXI_Change_Chan

Continued

However, you will need to call `SCXI_Single_Chan_Setup` again if you want to select a channel on a different module.

SCXI_Configure_Filter

Format

status = SCXI_Configure_Filter (chassisID, moduleSlot, channel, filterMode, freq, cutoffDivDown, outClkDivDown, actualFreq)

Purpose

Configures the filter on any SCXI module that supports programmable filter settings. Currently, only the SCXI-1122 and SCXI-1141 have programmable filter settings; the other analog input modules have hardware-selectable filters.

Parameters

Input

Name	Type		Description
	Windows	Windows NT	
chassisID	I16	I32	chassis ID number
moduleSlot	I16	I32	chassis slot number of the module
channel	I16	I32	module channel
filterMode	I16	I32	filter configuration mode
freq	F64	F64	filter cutoff frequency
cutoffDivDown	U16	U32	external signal divisor for cutoff frequency
outClkDivDown	U16	U32	clock signal divisor to send to OUTCLK

SCXI_Configure_Filter

Continued

Output

Name	Type		Description
	Windows	Windows NT	
actualFreq	F64	F64	actual filter cutoff frequency

Parameter Discussion

channel is the module channel for which you want to change the filter configuration. If **channel** = -1, `SCXI_Configure_Filter` changes the filter configuration for all channels on the module.

filterMode indicates the filter configuration mode for the given **channel**.

0:	Bypass the filter.
1:	Set filter cutoff frequency to **freq**.
2:	Configure the filter to use an external signal. The module divides the external signal by **cutoffDivDown** to determine the filter cutoff frequency. The module also divides the external signal by **outClkDivDown** and sends it to the module front connector OUTCLK pin. You can use this filter mode to configure a tracking filter. You can only use this mode with the SCXI-1141.
3:	Enable the filter (the reverse of **filterMode** 0).

freq is the cutoff frequency you want to select from the frequencies available on the module if **filterMode** = 1.

The SCXI-1122 has two possible cutoff frequencies:

4.0:	-10 dB at 4 Hz
4,000.0:	-3 dB at 4 kHz

The SCXI-1141 has a range of cutoff frequencies from 10 Hz to 25 kHz. `SCXI_Configure_Filter` produces the frequency you want as closely as possible by dividing an internal 10 MHz signal on the SCXI-1141. The function returns the exact cutoff frequency produced in the output parameter **actualFreq**.

If **filterMode** = 2, set **freq** to the *approximate* frequency of the external signal you are using. Chapter 2 of the *SCXI-1141 User Manual* explains the impact of different signal frequencies on the filters.

SCXI_Configure_Filter

Continued

If **filterMode** = 0 or 3, NI-DAQ ignores **freq**.

cutoffDivDown is an integer by which the module divides the external signal to determine the filter cutoff frequency when **filterMode** = 2. NI-DAQ ignores this parameter if **filterMode** is not 2.
Range: 2 to 65,535

outClkDivDown is an integer by which the module divides either the internal 10 MHz signal (if **filterMode** = 1) or the external signal (if **filterMode** = 2) to send back to the module front connector OUTCLK pin. This parameter is only used for the SCXI-1141.
Range: 2 to 65,535

actualFreq returns the actual cutoff frequency that the module uses.

Using this Function

The SCXI-1122 has one filter setting applied to all channels on the module; therefore, you must set **channel** = -1. The SCXI-1122 only works with **filterMode** = 1; you cannot configure the SCXI-1122 to bypass the filter or to use an external signal to set the cutoff frequency. The default frequency setting for the SCXI-1122 is 4 Hz.

The SCXI-1141 also has one filter setting applied to all channels, so you must use **channel** = -1 when you select a cutoff frequency for that module. After you select the cutoff frequency for the entire module, you can configure one or more of the channels to enable the filter by calling `SCXI_Configure_Filter` again for each channel and setting **filterMode** = 3. By default, all the channel filters on the SCXI-1141 are bypassed.

SCXI_Get_Chassis_Info

Format

status = SCXI_Get_Chassis_Info(SCXIchassisID, chassisType, chassisAddress, commMode, commPath, numSlots)

Purpose

Returns chassis configuration information.

Parameters

Input

Name	**Type**		**Description**
	Windows	**Windows NT**	
SCXIchassisID	I16	I32	logical ID assigned to the SCXI chassis

Output

Name	**Type**		**Description**
	Windows	**Windows NT**	
chassisType	I16	I32	type of SCXI chassis
chassisAddress	I16	I32	hardware jumpered address of an SCXI-1001 chassis
commMode	I16	I32	communication mode
commPath	I16	I32	communication path
numSlots	I16	I32	number of plug-in module slots

SCXI_Get_Chassis_Info

Continued

Parameter Discussion

chassisType indicates what type of SCXI chassis is configured for the given **SCXIchassisID**.

0:	SCXI-1000 4-slot chassis.
1:	SCXI-1001 12-slot chassis.
2:	SCXI-2000 4-slot remote chassis.

chassisAddress is the hardware-jumpered address of an SCXI chassis.

Range:	0 to 31 (SCXI-1000, SCXI-1001).
	0 to 255 (SCXI-2000 or SCXI chassis with SCXI-2400 module).

commMode is the Communication mode that will be used when the driver communicates with the SCXI chassis and modules.

0:	Communication mode is disabled. In effect, the chassis is disabled.
1:	Enables serial communication through a digital output port of a DAQ device that is cabled to a module in the chassis.
2:	Enables parallel communication over the PC parallel port that is cabled to the SCXI-1200 module.
3:	Enables serial communication over the PC serial port that is cabled to one or more SCXI-2000 chassis or SCXI-2400 modules.

commPath is the communication path that will be used when the driver communicates with the SCXI chassis and modules. If **commMode** = 1 or 2, **commPath** should be the device number of the DAQ device that is the designated communicator for the chassis. If **commMode** = 3, **commPath** is the serial port for this chassis. When **commMode** = 0, **commPath** is meaningless.

numSlots is the number of plug-in module slots in the SCXI chassis.

4:	For the SCXI-1000 and SCXI-2000 chassis.
12:	For the SCXI-1001 chassis.

☞ **Note:** ***C Programmers*—chassisType, chassisAddress, commMode, commPath,** ***and* numSlots *are pass-by-reference parameters.***

SCXI_Get_Module_Info

Format

status = SCXI_Get_Module_Info (SCXIchassisID, moduleSlot, modulePresent, operatingMode, DAQdeviceNumber)

Purpose

Returns configuration information for the given chassis slot number.

Parameters

Input

Name	Type		Description
	Windows	Windows NT	
SCXIchassisID	I16	I32	logical ID assigned to the SCXI chassis
moduleSlot	I16	I32	chassis slot number

Output

Name	Type		Description
	Windows	Windows NT	
modulePresent	I32	I32	type of module present in given slot
operatingMode	I16	I32	multiplexed or parallel mode
DAQdeviceNumber	I16	I32	device number of the DAQ device that is cabled to the module

SCXI_Get_Module_Info

Continued

Parameter Discussion

modulePresent indicates what type of module is present in the given slot.

-1:	Empty slot; there is no module present in the given slot.
2:	SCXI-1121.
4:	SCXI-1120.
6:	SCXI-1100.
8:	SCXI-1140.
10:	SCXI-1122.
12:	SCXI-1160.
14:	SCXI-1161.
16:	SCXI-1162.
18:	SCXI-1163.
20:	SCXI-1124.
24:	SCXI-1162HV.
28:	SCXI-1163R.
30:	SCXI-1102.
32:	SCXI-1141.
38:	SCXI-1200.
40:	SCXI-2400.

Any other value returned in the **modulePresent** parameter indicates that an unfamiliar module is present in the given slot.

operatingMode indicates whether the module present in the given slot is being operated in Multiplexed or Parallel mode. Please refer to Chapter 15, *SCXI Hardware*, in the *DAQ Hardware Overview Guide* for an explanation of each operating mode. If the slot is empty, **operatingMode** is meaningless.

0:	Multiplexed operating mode.
1:	Parallel operating mode.

DAQdeviceNumber is the device number of the DAQ device in the PC that is cabled directly to the module present in the given slot. If the slot is empty, **DAQdeviceNumber** is meaningless.

0:	No DAQ device is cabled to the module.
n,	where *n* is the device number of the DAQ device cabled to the module.

If the **moduleSlot** contains an SCXI-1200, **DAQdeviceNumber** is the logical device number of the SCXI-1200. If the **moduleSlot** contains an SCXI-2400, **DAQdeviceNumber** is 0.

SCXI_Get_Module_Info

Continued

☞ **Note:** ***C Programmers*—modulePresent, operatingMode, *and* DAQdeviceNumber *are pass-by-reference parameters*.**

SCXI_Get_State

Format

status = SCXI_Get_State (SCXIChassisID, moduleSlot, port, channel, data)

Purpose

Gets the state of a single channel or an entire port on a digital or relay SCXI module.

Parameters

Input

Name	Type		Description
	Windows	Windows NT	
SCXIChassisID	I16	I32	chassis ID number
moduleSlot	I16	I32	module slot number
port	I16	I32	port of the module to write to (all current modules support only Port 0)
channel	I16	I32	channel of the specified port to read from

Output

Name	Type		Description
	Windows	Windows NT	
data	U32	U32	Contains data read from a single channel or a digital pattern for an entire port

SCXI_Get_State

Continued

Parameter Discussion

port is the port number of the module to be read from. Currently, all of the SCXI modules support only Port 0.

channel is the channel number on the specified port.

n:	Read from a single channel.
	SCXI-1160: $0 \leq n < 16$.
	SCXI-1161: $0 \leq n < 8$.
	SCXI-1162: $0 \leq n < 32$.
	SCXI-1162HV: $0 \leq n < 32$.
	SCXI-1163: $0 \leq n < 32$.
	SCXI-1163R: $0 \leq n < 32$.
-1:	Read the state pattern from an entire port.

When **channel** = -1, **data** contains the pattern of an entire port. Bit 0 corresponds to the state of channel 0 in the port, and the states of the other channels are represented in ascending order in **data** so that bit n corresponds to channel n. If the port is less than 32 bits wide, the unused bits in **data** are set to zero.

When **channel** = n, the LSB (bit 0) of **data** contains the state of channel n on the specified port.

For relay modules, a 0 bit indicates that the relay is closed or in the normally closed position, and a 1 indicates that the module is open or in the normally open position. For SCXI digital modules, a 0 bit indicates that the line is low, and a 1 bit indicates that the line is high.

☞ **Note:** ***For a discussion of the NC and NO positions, please see your SCXI module user manual.***

☞ **Note:** ***C Programmers—*****data** ***is a pass-by-reference parameter.***

Using This Function

The SCXI-1160 is a latching module; in other words, the module powers up with its relays in the position they were left at power down. Thus, at the beginning of an NI-DAQ application, there is no way to know the states of the relays. The driver will retain the state of a relay as soon as a hardware write takes place.

The SCXI-1161 is a nonlatching module and powers up with its relays in the NC position. After you call `SCXI_Load_Config` or `SCXI_Set_Config`, an actual

SCXI_Get_State

Continued

hardware write to the relays must take place before the driver can obtain the state information of the relays, just like the SCXI-1160. You can call `SCXI_Reset` to do this.

The SCXI-1163 and 1163R are optocoupler output modules with 32 digital output channels and 32 solid state relay channels, respectively. NI-DAQ can read the states of the module only if the module is jumper configured and operating in Parallel mode. When operating in Serial or Multiplexed mode, the driver retains the states of the digital output lines in memory. Consequently, a hardware write must take place before the driver can obtain the states of the module.

You should call `SCXI_Reset` after a call to `SCXI_Set_Config` or `SCXI_Load_Config` for the SCXI-1160, SCXI-1161, SCXI-1163, and SCXI-1163R modules.

Remember that only on the SCXI-1162, SCXI-1163, SCXI-1162HV, and SCXI-1163R in Parallel mode does NI-DAQ read the states from hardware. On both the SCXI-1160 and SCXI-1161, the driver keeps a software copy of the relay states in memory.

SCXI_Get_Status

Format

status = SCXI_Get_Status (SCXIChassisID, moduleSlot, wait, data)

Purpose

Reads the data in the Status Register on the specified module. This function supports the SCXI-1160, SCXI-1102, SCXI-1122, and SCXI-1124 modules.

Parameters

Input

Name	Type		Description
	Windows	Windows NT	
SCXIChassisID	I16	I32	chassis ID number
moduleSlot	I16	I32	module slot number
wait	I16	I32	determines if the function should poll the Status Register, until timeout, for the SCXI module to become ready

Output

Name	Type		Description
	Windows	Windows NT	
data	U32	U32	contains the contents of the Status Register

SCXI_Get_Status

Continued

Parameter Discussion

wait determines if the function should poll the Status Register on the module until either the module is ready or timeout is reached. If the module is not ready by timeout, NI-DAQ returns a timeout error.

1: The function will poll the Status Register on the module, until ready or timeout.

0: The function will read and return the Status Register on the module.

data contains the contents of the Status Register.

0: Indicates that the module is busy. Do not perform any further operations on the modules until the status bit goes high again. This means the SCXI-1122 or SCXI-1160 relays are still switching or the SCXI-1124 DACs are still settling.

1: Indicates that the module is ready. The SCXI-1122 or SCXI-1160 relays are finished switching or the SCXI-1124 DACs have settled.

☞ **Note:** ***C Programmers*—data *is a pass-by-reference parameter.***

Using This Function

If **wait** = 1, the function will wait a maximum of 100 ms for the module status to be ready. If, while polling the Status Register, a timeout occurs, the output parameter **data** returns the current value of the Status Register.

The SCXI-1160, SCXI-1102, SCXI-1122, and SCXI-1124 Status Registers contain only one bit, so only the least significant bit of the data parameter is meaningful.

SCXI_Load_Config

Format

status = SCXI_Load_Config (SCXIchassisID)

Purpose

Loads the SCXI chassis configuration information that you established in the configuration utility. Sets the software states of the chassis and the modules present to their default states. This function makes no changes to the hardware state of the SCXI chassis or modules.

Parameters

Input

Name	**Type**		**Description**
	Windows	**Windows NT**	
SCXIchassisID	I16	I32	logical ID assigned to the SCXI chassis

Using This Function

It is important to realize that this function makes no changes to the hardware. To reset the hardware to its default state, you should use the `SCXI_Reset` function. Refer to the `SCXI_Reset` function description for a listing of the default states of the chassis and modules.

It is possible to programmatically change the configuration that you established in the configuration utility using the `SCXI_Set_Config` function.

SCXI_MuxCtr_Setup

Format

status = SCXI_MuxCtr_Setup (deviceNumber, enable, scanDiv, ctrValue)

Purpose

Enables or disables a DAQ device counter to be used as a multiplexer counter during SCXI channel scanning to synchronize the MIO or AI device, Lab-PC-1200, Lab-PC-1200AI, Lab-PC+, or SCXI-1200 scan list with the module scan list that NI-DAQ has downloaded to Slot 0 of the SCXI chassis.

Parameters

Input

Name	Type		Description
	Windows	Windows NT	
deviceNumber	I16	I32	assigned by configuration utility
enable	I16	I32	whether to enable counter 1 to be a mux counter
scanDiv	I16	I32	whether the mux counter will divide the scan clock
ctrValue	U16	U32	value to be programmed into the mux counter

Parameter Discussion

enable indicates whether to enable a device counter to be a mux counter for subsequent SCXI channel scanning operations.

0:	Disable the mux counter; the device counter is freed.
1:	Enable the device counter to be a mux counter.

SCXI_MuxCtr_Setup

Continued

scanDiv indicates whether the mux counter will divide the scan clock during the acquisition.

0:	The mux counter does not divide the scan clock; it simply pulses after every *n* mux-gain entry on the DAQ device, where *n* is the **ctrValue**. The mux counter pulses are currently not used by the SCXI chassis or modules, so this mode is not useful.
1:	The mux counter divides the scan clock so that *n* conversions are performed for every mux-gain entry on the DAQ device, where *n* is the **ctrValue**.

ctrValue is the value NI-DAQ will program into the mux counter. If **enable** = 1 and **scanDiv** =1, **ctrValue** is the number of conversions NI-DAQ will perform on each mux-gain entry on the DAQ device. If **enable** = 0, NI-DAQ ignores this parameter.

Using This Function

You can use this function to synchronize the scan list that NI-DAQ has loaded into the mux-gain memory of the DAQ device and the SCXI module scan list that NI-DAQ has loaded into Slot 0 of the SCXI chassis. The total number of samples to be taken in one pass through each scan list should be the same. Am9513-based devices use counter 1 as the mux counter. The Lab-PC-1200, Lab-PC-1200AI, Lab-PC+, SCXI-1200, and E Series devices have a dedicated mux counter.

For example, for the following scan lists, a **ctrValue** of 8 would cause NI-DAQ to take eight samples for each MIO or AI scan list entry. The first two entries in the module scan list will occur during the first entry of the MIO or AI scan list, at an MIO or AI gain of 5. The third module scan list entry will occur during the second entry of the MIO or AI scan list, at an MIO or AI gain of 10. Thus, NI-DAQ uses the **ctrValue** here to distribute different MIO or AI gains across the module scan list, as well as to make the scan list lengths equal at 16 samples each.

Am9513-based devices use counter 1 as the mux counter. The Lab-PC-1200, Lab-PC-1200AI, Lab-PC+, SCXI-1200, and E Series devices have a dedicated mux counter.

SCXI_MuxCtr_Setup

Continued

Module Scan List		MIO or AI Scan List	
Module	**Number of Samples**	**Channel**	**Gain**
2	4	0	5
3	4	0	10
4	8	—	—

Another example would use the same module scan list as above, but use an MIO or AI scan list with only one entry for channel 0. In this case, a **ctrValue** of 16 would be appropriate.

SCXI_Reset

Format

status = SCXI_Reset (SCXIchassisID, moduleSlot)

Purpose

Resets the specified module to its default state. You can also use `SCXI_Reset` to reset the Slot 0 scanning circuitry or to reset the entire chassis.

Parameters

Input

Name	Type		Description
	Windows	Windows NT	
SCXIchassisID	I16	I32	logical ID assigned to the SCXI chassis
moduleSlot	I16	I32	chassis slot number of the module

Parameter Discussion

moduleSlot is the chassis slot number of the module that is to be reset.

Range:	1 to *n*, where *n* is the number of slots in the chassis.
0:	Reset Slot 0 of the chassis by resetting the module scan list and scanning circuitry. If this is a remote SCXI chassis, Slot 0 is rebooted and it will take a few seconds for this call to return because it waits for the chassis to finish booting and attempts to reestablish communication with the chassis.
-1:	Reset all modules present in the chassis and reset Slot 0.

Using This Function

The default states of the SCXI modules are as follows:

- SCXI-1100 and SCXI-1122:
  Module gain = 1.
  Module filter = 4 Hz (SCXI-1122 only).
  Channel 0 is selected.
  Multiplexed channel scanning is disabled.

SCXI_Reset

Continued

Module output is enabled if the module is cabled to a DAQ device.
Calibration is disabled.

- SCXI-1120, SCXI-1121, and SCXI-1140:
  - If the module is operating in Multiplexed mode:
    Channel 0 is selected.
    Multiplexed channel scanning is disabled.
    Module output is enabled if the module is cabled to a DAQ device.
    Hold count is 1.
  - If the module is operating in Parallel mode:
    All channels are enabled.
    Track/hold signal is disabled.
- SCXI-1124:
  Sets the voltage range for each channel to 0–10 V. Writes a binary 0 to each DAC.
- SCXI-1141:
  - If the module is in Multiplexed mode:
    Channel 0 is selected.
    Amplifier gains = 1.
    Filters are bypassed.
    MUXed scanning is disabled.
    Module output is enabled if module is cabled to a DAQ device.
    Autozeroing is disabled.
  - If the module is in Parallel mode:
    All channels are enabled.
    Amplifier gains = 1.
    Filters are bypassed.
    Autozeroing is disabled.
- SCXI-1160:
  Sets the current state information of relays in memory to unknown. No hardware write takes place.
- SCXI-1161:
  Initializes all of the relays on the module to the Normally Closed position. It also updates the software copy of the status maintained by the driver.
- SCXI-1163:
  Initializes all of the digital output lines on the module to a logical high state.

SCXI_Reset

Continued

- SCXI-1163R:
  Initializes all of the solid state relays to their open states.
- SCXI-1200:
  Sets channel 0 to read from the front panel 50-pin connector and not the SCXI bus. Use `Init_DA_Brds` to completely initialize the hardware and software state of the SCXI-1200.
- SCXI-2400:
  Reboots the module. It will take a few seconds for this call to return because it waits for the module to finish booting and attempts to reestablish communication with the module.

SCXI_Scale

Format

status = SCXI_Scale (SCXIchassisID, moduleSlot, channel, SCXIgain, TBgain, DAQboard, DAQchannel, DAQgain, numPoints, binArray, voltArray)

Purpose

Scales an array of binary data acquired from an SCXI channel to voltage. `SCXI_Scale` uses stored software calibration constants if applicable for the given module when it scales the data. The SCXI-1122 and SCXI-1141 have default software calibration constants loaded from the module EEPROM; all other analog input modules have no software calibration constants unless you follow the analog input calibration procedure outlined in the `SCXI_Cal_Constants` function description.

☞ **Note:** ***NI-DAQ for Windows NT does not support this function.***

Parameters

Input

Name	Type		Description
	Windows	Windows NT	
SCXIchassisID	I16	n/a	SCXI chassis ID number
moduleSlot	I16	n/a	SCXI module slot number
channel	I16	n/a	SCXI channel from which the data was acquired
SCXIgain	F64	n/a	SCXI gain setting for the channel
TBgain	F64	n/a	gain applied at SCXI terminal block, if any
DAQboard	I16	n/a	device number of the DAQ device that acquired the data

SCXI_Scale

Continued

Name	Type		Description
	Windows	Windows NT	
DAQchannel	I16	n/a	onboard DAQ channel used in the acquisition
DAQgain	I16	n/a	DAQ device gain used in the acquisition
numPoints	U32	n/a	number of data points to scale
binArray	[I16]	n/a	binary data returned from acquisition

Output

Name	Type		Description
	Windows	Windows NT	
voltArray	[F64]	n/a	array of scaled data

Parameter Discussion

channel is the number of the channel on the SCXI module.

Range:	0 to *n*-1, where *n* is the number of channels available on the module.
-1:	Scale data acquired from the temperature sensor on the terminal block connected to the module if the temp. sensor is in the MTEMP configuration.

SCXIgain is the SCXI module or channel gain setting. Valid **SCXIgain** values depend on the module type:

SCXI-1100:	1, 2, 5, 10, 20, 50, 100, 200, 500, 1,000, 2,000.
SCXI-1120:	1, 2, 5, 10, 20, 50, 100, 200, 250, 500, 1,000, 2,000.
SCXI-1121:	1, 2, 5, 10, 20, 50, 100, 200, 250, 500, 1,000, 2,000.
SCXI-1122:	0.01, 0.02, 0.05, 0.1, 0.2, 0.5, 1, 2, 5, 10, 20, 50, 100, 200, 500, 1,000, 2,000.
SCXI-1140:	1, 10, 100, 200, 500.
SCXI-1141:	1, 2, 5, 10, 20, 50, 100.

SCXI_Scale

Continued

TBgain is the gain applied at the SCXI terminal block. Currently, only the SCXI-1327 terminal block can apply gain to your SCXI module channels; it has DIP switches to choose a gain of 1.0 or 0.01 for each input channel. You can use the SCXI-1327 with the SCXI-1120 and SCXI-1121 modules. For terminal blocks that do not apply gain to your SCXI channels, set **TBgain** = 1.0.

DAQboard is the device number of the DAQ device you used to acquire the binary data. This should be the same device number that you passed to the `DAQ` or `SCAN` function call, and the same **DAQboard** number you passed to `SCXI_Single_Chan_Setup` or `SCXI_SCAN_Setup`.

DAQchannel is the DAQ device channel number you used to acquire the binary data. This should be the same channel number that you passed to the `DAQ` or `SCAN` function call. For most cases, you will be multiplexing all of your SCXI channels into DAQ device channel 0.

DAQgain is the DAQ device gain you used to acquire the binary data. This should be the same gain code that you passed to the `DAQ` or `SCAN` function call. For most cases, you will use a DAQ device gain of 1, and you will set any gain you need at the SCXI module.

numPoints is the number of data points you want to scale for the given channel. The **binArray** and **voltArray** parameters must be arrays of length **numPoints** (at least). If you acquired data from more than one SCXI channel, you must be careful to pass the number of points for this channel only, not the total number of points you acquired from *all* channels.

binArray is the array of binary data for the given channel. **binArray** should contain **numPoints** data samples from the SCXI **channel**. If you acquired data from more than one SCXI channel, you need to demultiplex the binary data that was returned from the `SCAN` call before you call `SCXI_Scale`. You can use the `SCAN_Demux` call to do this. After demuxing the binary data, you should call `SCXI_Scale` once for *each* SCXI channel, passing in the appropriate demuxed binary data for each channel.

voltArray is the output array for the scaled voltage data. **voltArray** should be at least **numPoints** elements long.

SCXI_Scale

Continued

Using This Function

`SCXI_Scale` uses the following equation to scale the binary data to voltage:

$$\mathbf{voltArray}[\mathrm{i}] = \frac{(\mathbf{binArray}[\mathbf{i}] - \mathit{binaryOffset})(\mathit{voltageResolution})}{(\mathbf{SCXIgain})(\mathbf{TBgain})(\mathbf{DAQgain})(\mathit{gainAdjust})}$$

The ***voltage resolution*** depends on your DAQ device and its range and polarity settings. For example, the AT-MIO-16 in bipolar mode with an input range of -10 to 10 V has a voltage resolution of 4.88 mV per LSB.

NI-DAQ automatically loads *binaryOffset* and *gainAdjust* for the SCXI-1122 for all of its gain settings from the module EEPROM. The SCXI-1122 module is shipped with factory calibration constants for *binaryOffset* and *gainAdjust* loaded in the EEPROM. You can calculate your own calibration constants and store them in the EEPROM and in NI-DAQ memory for `SCXI_Scale` to use. Please refer to the procedure outlined in the `SCXI_Cal_Constants` function description. The same is true for the SCXI-1141, except binaryOffset is not on the SCXI-1141 EEPROM and defaults to 0.0. However, you can calculate your own *binaryOffset* using the procedure outlined in the `SCXI_Cal_Constants` function description.

For other analog input modules, *binaryOffset* defaults to 0.0 and *gainAdjust* defaults to 1.0. However, you can calculate your own calibration constants and store them in NI-DAQ memory for NI-DAQ to use in the `SCXI_Scale` function by following the procedure outlined in the `SCXI_Cal_Constants` function description.

SCXI_SCAN_Setup

Format

status = SCXI_SCAN_Setup (SCXIchassisID, numModules, moduleList, numChans, startChans, DAQdeviceNumber, modeFlag)

Purpose

Sets up the SCXI chassis for a multiplexed scanning data acquisition to be performed by the given DAQ device. You can scan modules in any order; however, you must scan channels on each module in consecutive order. The function downloads a module scan list to Slot 0 in the SCXI chassis that will determine the sequence of modules to be scanned and how many channels on each module NI-DAQ will scan. NI-DAQ programs each module with its given start channel and resolves any contention on the SCXIbus.

Parameters

Input

Name	Type		Description
	Windows	Windows NT	
SCXIchassisID	I16	I32	logical ID assigned to the SCXI chassis
numModules	I16	I32	number of modules to be scanned
moduleList	[I16]	[I32]	list of module slot numbers
numChans	[I16]	[I32]	how many channels to scan on each module
startChans	[I16]	[I32]	contains the start channels for each module
DAQdeviceNumber	I16	I32	the DAQ device that will be performing the channel scanning
modeFlag	I16	I32	scanning mode to be used

SCXI_SCAN_Setup

Continued

Parameter Discussion

numModules is the number of modules to be scanned, and the length of the **moduleList**, **numChans**, and **startChans** arrays.
Range: 1 to 256.

moduleList is an array of length **numModules** containing the list of module slot numbers corresponding to the modules to be scanned.
Range: **moduleList**[i] =1 to *n*, where *n* is the number of slots in the chassis.

Any value in the **moduleList** array that is greater than the number of slots available in the chassis (such as a value of 15 or 16) can act as a dummy entry in the module scan list. Dummy entries are very useful in multichassis scanning operations to indicate in the module scan list when the MIO or AI is scanning channels on another chassis.

numChans is an array of length **numModules** that indicates how many channels to scan on each module represented in the **moduleList** array. If the number of channels specified for a module exceeds the number of input channels available on the module, the channel scanning will wrap around after the last input channel and continue with the first input channel. If a module is represented more than once in the **moduleList** array, there can be different **numChans** values for each entry. For the SCXI-1200, this parameter depends entirely on its corresponding **startChans** value.
Range: **numChans**[i] = 1 to 128.

startChans is an array of length **numModules** that contains the start channels for each module represented in the **moduleList** array. If a module is represented more than once in the **moduleList** array, the corresponding elements in the **startChans** array should contain the same value; *there can only be one start channel for each module*.

startChans[i] = 0 to n-1, where n is the number of input channels available on the corresponding module, selects the indicated channel as the lowest scanned channel. NI-DAQ will scan a total of **numModules** successive channels starting with this channel.

(SCXI-1102 only)—**startChans**[i] = c + `ND_CJ_TEMP`, where c is a channel number as described above, selects scanning of the temperature sensor on the terminal block, followed by successive channels beginning with c. NI-DAQ will scan the temperature sensor and then a total of **numModules**-1 successive channels starting with this channel, for a total of **numModules** total readings.

startChans[i] = -1 selects only the temperature sensor on the terminal block; no channels are scanned.

SCXI_SCAN_Setup

Continued

Note that if you use -1 to select the temperature sensor, all readings from that module will be readings of the temperature sensor only; channel scanning is not possible.

DAQdeviceNumber is the device number of the DAQ device that will perform the channel scanning operation. If you are using the SCXI-1200 to perform the data acquisition, you should specify the module logical device number.

modeFlag indicates the scanning mode to be used. Only one scanning mode is currently supported, so you should always set this parameter to zero.

SCXI_Set_Config

Format

status = SCXI_Set_Config (SCXIchassisID, chassisType, chassisAddress, commMode, commPath, numSlots, modulesPresent, operatingModes, connectionMap)

Purpose

Changes the software configuration of the SCXI chassis that you established in the configuration utility. Sets the software states of the chassis and the modules specified to their default states. This function makes no changes to the hardware state of the SCXI chassis or modules.

☞ **Note:** ***You cannot use this function to configure a chassis that contains an SCXI-1200.***

Parameters

Input

Name	Type		Description
	Windows	Windows NT	
SCXIchassisID	I16	I32	logical ID assigned to the SCXI chassis
chassisType	I16	I32	type of SCXI chassis
chassisAddress	I16	I32	hardware-jumpered address
commMode	I16	I32	communication mode used
commPath	I16	I32	communication path used
numSlots	I16	I32	number of plug-in module slots
modulesPresent	[I32]	[I32]	type of module present in each slot

SCXI_Set_Config

Continued

Name	Type		Description
	Windows	Windows NT	
operatingModes	[I16]	[I32]	the operating mode of each module
connectionMap	[I16]	[I32]	describes the connections between the SCXI chassis and the DAQ devices

Parameter Discussion

chassisType indicates what type of SCXI chassis is configured for the given **SCXIchassisID**.

0: SCXI-1000 4-slot chassis.
1: SCXI-1001 12-slot chassis.

chassisAddress is the hardware jumpered address of an SCXI chassis.
Range: 0 to 31.

commMode is the communication mode that will be used when the driver communicates with the SCXI chassis and modules.

0: Communication mode is disabled. In effect, this disables the chassis.
1: Enables serial communication through a digital output port of a DAQ device that is cabled to a module in the chassis.
2: Enables serial communication through the parallel port cabled to an SCXI-1200 in the chassis.

commPath is the communication path that will be used when the driver communicates with the SCXI chassis and modules. When **commMode** = 1 or 2, set the path to the device number of the DAQ device that is the designated communicator for the chassis. If only one DAQ device is connected to the chassis, set **commPath** to the device number of that device. If more than one DAQ device is connected to modules in the chassis, you must designate one device as the communicator device, and you should set its device number to **commPath**. Refer to the **connectionMap** array description; you should set **commPath** to one of the device numbers specified in that array. When **commMode** = 0 or 3, NI-DAQ ignores **commPath**.

numSlots is the number of plug-in module slots in the SCXI chassis.

SCXI_Set_Config

Continued

4:	For the SCXI-1000 chassis.
12:	For the SCXI-1001 chassis.

modulesPresent is an array of length **numSlots** that indicates what type of module is present in each slot. The first element of the array corresponds to slot 1 of the chassis, and so on.

-1:	Empty slot; there is no module present in the corresponding slot.
2:	SCXI-1121.
4:	SCXI-1120.
6:	SCXI-1100.
8:	SCXI-1140.
10:	SCXI-1122.
12:	SCXI-1160.
14:	SCXI-1161.
16:	SCXI-1162.
18:	SCXI-1163.
20:	SCXI-1124.
24:	SCXI-1162HV.
28:	SCXI-1163R.
30:	SCXI-1102.
32:	SCXI-1141.

Any other value for an element of the **modulesPresent** array indicates that a module that is unfamiliar to NI-DAQ (such as a custom-built module) is present in the corresponding slot.

operatingModes is an array of length **numSlots** that indicates the operating mode of each module in the **modulesPresent** array—multiplexed or parallel. Please refer to Chapter 15, *SCXI Hardware*, of the *DAQ Hardware Overview Guide* for an explanation of each operating mode. If any of the slots are empty (indicated by a value of -1 in the corresponding element of the **modulesPresent** array), NI-DAQ ignores the corresponding element in the **operatingModes** array.

0:	Multiplexed operating mode.
1:	Parallel operating mode.
2:	Parallel operating mode using the secondary connector of the DAQ device.

connectionMap is an array of length **numSlots** that describes the connections between the SCXI chassis and the DAQ devices in the PC. For each module present in the chassis, you must specify the device number of the DAQ device that is cabled to the module, if there is one. For the SCXI-1200 module, you should specify the logical device number

SCXI_Set_Config

Continued

of the module. If any of the slots are empty (indicated by a value of -1 in the corresponding element of the **modulesPresent** array), NI-DAQ ignores the corresponding element of the **connectionMap** array. The **commPath** parameter value must be one of the DAQ device numbers specified in this array.

0:	No DAQ device is cabled to the module.
n,	where *n* is the device number of the DAQ device cabled to the module.

Using This Function

The configuration information that was saved to disk by the configuration utility will remain unchanged; this function changes only the configuration in the current application. Any subsequent calls to `SCXI_Load_Config` will reload the configuration from the configuration utility.

Remember, the hardware state of the chassis is not affected by this function; you should use the `SCXI_Reset` function to reset the hardware states. Refer to the `SCXI_Reset` function description for a listing of the default states of the chassis and modules.

SCXI_Set_Gain

Format

status = SCXI_Set_Gain (SCXIchassisID, moduleSlot, channel, gain)

Purpose

Sets the specified channel to the given gain setting on any SCXI module that supports programmable gain settings. Currently, the SCXI-1100, SCXI-1122, and SCXI-1141 have programmable gains; the other analog input modules have hardware-selectable gains.

Parameters

Input

Name	Type		Description
	Windows	Windows NT	
SCXIchassisID	I16	I32	chassis ID number
moduleSlot	I16	I32	module slot number
channel	I16	I32	module channel
gain	F64	F64	gain setting

Parameter Discussion

channel is the module channel you want to change the gain setting for. If **channel** = -1, `SCXI_Set_Gain` changes the gain for all channels on the module. The SCXI-1100 and SCXI-1122 have one gain amplifier, so all channels have the same gain setting; therefore, you must set **channel** = -1 for those modules.

gain is the gain setting you want to use. Notice that **gain** is a double-precision floating point parameter. Valid gain settings depend on the module type:

SCXI-1100:	1, 2, 5, 10, 20, 50, 100, 200, 500, 1,000, 2,000.
SCXI-1122:	0.01, 0.02, 0.05, 0.1, 0.2, 0.5, 1, 2, 5, 10, 20, 50, 100, 200, 500, 1,000, 2,000.
SCXI-1141:	1, 2, 5, 10, 20, 50, 100.

SCXI_Set_Input_Mode

Format

status = SCXI_Set_Input_Mode (SCXIchassisID, moduleSlot, inputMode)

Purpose

Configures the SCXI-1122 channels for two-wire mode or four-wire mode.

☞ **Note:** ***NI-DAQ for Windows NT does not support this function.***

Parameters

Input

Name	Type		Description
	Windows	Windows NT	
SCXIchassisID	I16	n/a	chassis ID number
moduleSlot	I16	n/a	module slot number
inputMode	I16	n/a	channel input mode configuration

Parameter Discussion

inputMode is the channel configuration you want to use.

0: two-wire mode (module default).
1: four-wire mode.

Using This Function

When the SCXI-1122 is in two-wire mode (module default setting), the module is configured for 16 differential input channels.

When the SCXI-1122 is in four-wire mode, channels 0 through 7 are configured to be differential input channels, and channels 8 through 15 are configured to be current excitation channels. The SCXI-1122 has a current excitation source that will switch to drive the corresponding excitation channel 8 through 15 whenever you select an input channel 0 through 7. Channel 8 will produce the excitation when you select input channel 0, channel 9 will produce the excitation when you select input channel 1, and so

SCXI_Set_Input_Mode

Continued

on. You can use four-wire mode for single point data acquisition, or for multiple channel scanning acquisitions. During a multiple channel scan, the excitation channels will switch simultaneously with the input channels.

You can hook up an RTD or thermistor to your input channel that uses the corresponding excitation channel to drive the transducer.

You can call the `SCXI_Set_Input_Mode` function to enable 4-wire mode at any time before you start the acquisition; you can call `SCXI_Set_Input_Mode` again after the acquisition to return the module to normal two-wire mode.

SCXI_Set_State

Format

status = SCXI_Set_State (SCXIChassisID, module, port, channel, data)

Purpose

Sets the state of a single channel or an entire port on a digital output or relay module.

Parameters

Input

Name	Type		Description
	Windows	Windows NT	
SCXIChassisID	I16	I32	chassis ID number
module	I16	I32	module slot number
port	I16	I32	port of the module to write to (all current modules support only port 0)
channel	I16	I32	the channel on the specified port to change
data	U32	I32	contains new state information for a single channel or a digital pattern for an entire port

Parameter Discussion

port is the port number of the module to be written to. Currently, all of the SCXI modules support only port 0.

channel is the channel number on the specified port. Because all of the modules support only Port 0, **channel** maps to the actual channel on the module. If **channel** = -1, the function writes the pattern in **data** to the entire port.

n: Write to a single channel.
SCXI-1160: $0 \leq n < 16$.
SCXI-1161: $0 \leq n < 8$.

SCXI_Set_State

Continued

SCXI-1163: $0 \leq n < 32$.
SCXI-1163R: $0 \leq n < 32$.
-1: Write to an entire port.

When **channel** = -1, **data** contains the pattern of an entire port. Bit 0 corresponds to the state of channel 0 in the port, and the states of the other channels are represented in ascending order in **data** so that bit n corresponds to channel n. If the port is less than 32 bits wide, the unused bits in **data** are ignored.

When **channel** = n, the LSB (bit 0) of **data** contains the state of channel n on the specified port.

For relay modules, a 0 bit indicates that the relay is closed or in the normally closed position, and a 1 indicates that the module is open or in the normally open position. For SCXI digital modules, a 0 bit indicates that the line is low, and a 1 bit indicates that the line is high.

☞ **Note:** ***For a discussion of the NC and NO positions, please see your SCXI module user manual.***

Using This Function

Because the relays on the SCXI -1160 module have a finite lifetime, the driver will maintain a software copy of the relay states as you write to them; this allows the driver to excite the relays only when you specify a new relay state. If you call this function to specify the current relay state again, NI-DAQ will not excite the relay again. When the SCXI-1160 powers up, the relays remain in the same position as they were at power down. However, when you start an application, the driver does not know the states of the relays; it will excite all of the relays the first time you write to them and then remember the states for the remainder of the application. When you call the `SCXI_Reset` function, the driver will mark all relay states as unknown.

The SCXI-1161 powers up with its relays in the NC position. The SCXI-1163 powers up with its output lines high when you operate the module in multiplexed mode. The SCXI-1163R powers up with relays open. If you operate the SCXI-1163 or 1163R in parallel mode, the states of the output lines or relays are determined by the states of the corresponding lines on the DAQ device.

SCXI_Single_Chan_Setup

Format

status = SCXI_Single_Chan_Setup (SCXIchassisID, moduleSlot, moduleChan, DAQdeviceNumber)

Purpose

Sets up a multiplexed module for a single channel analog input operation to be performed by the given DAQ device. Sets the module channel, enables the module output, and routes the module output on the SCXIbus if necessary. Resolves any contention on the SCXIbus by disabling the output of any module that was previously driving the SCXIbus. You can also use this function to set up to read the temperature sensor on a terminal block connected to the front connector of the module.

Parameters

Input

Name	**Type**		**Description**
	Windows	**Windows NT**	
SCXIchassisID	I16	I32	logical ID assigned to the SCXI chassis
moduleSlot	I16	I32	chassis slot number
moduleChan	I16	I32	channel number of the input channel on the module
DAQdeviceNumber	I16	I32	device number of the DAQ device used to read the input channel

Parameter Discussion

moduleChan is the channel number of the input channel on the module that is to be read.

Range:	0 to *n*-1, where *n* is the number of input channels on the module.
-1:	Set up to read the temperature sensor on the terminal block connected to the module if the temperature sensor is in the MTEMP configuration.

SCXI_Single_Chan_Setup

Continued

DAQdeviceNumber is the device number of the DAQ device that will perform the analog input. If you will use the SCXI-1200 to perform the analog input, you should specify the module logical device number.

SCXI_Track_Hold_Control

Format

status = SCXI_Track_Hold_Control (SCXIchassisID, moduleSlot, state, DAQdeviceNumber)

Purpose

Controls the track/hold state of an SCXI-1140 module that you have set up for a single-channel operation.

☞ **Note:** ***This function is not supported for the E Series devices.***

Parameters

Input

Name	Type		Description
	Windows	Windows NT	
SCXIchassisID	I16	I32	logical ID assigned to the SCXI chassis
moduleSlot	I16	I32	chassis slot number
state	I16	I32	Track or hold mode
DAQdeviceNumber	I16	I32	device number of the DAQ device used to read the input channel

Parameter Discussion

moduleSlot is the chassis slot number of the SCXI-1140 module you want.
Range: 1 to *n*, where *n* is the number of slots in the chassis.

state indicates whether to put the module into track or hold mode.
0: Put the module into track mode.
1: Put the module into hold mode.

DAQdeviceNumber is the device number of the DAQ device that will perform the channel scanning operation. If you are using the SCXI-1200 to perform the data acquisition, you should specify the module logical device number.

SCXI_Track_Hold_Control

Continued

Using This Function

Please refer to the *SCXI Application Hints* discussion in Chapter 3, *Software Overview*, of the *NI-DAQ User Manual for PC Compatibles* for information about how to use the SCXI-1140 for single-channel and channel-scanning operations. This function is only needed for single-channel applications; the scan interval timer controls the track/hold state of the module during a channel-scanning operation. The *NI-DAQ User Manual for PC Compatibles* contains flowcharts for single-channel operations using the SCXI-1140 and this function.

SCXI_Track_Hold_Setup

Format

status = SCXI_Track_Hold_Setup (SCXIchassisID, moduleSlot, inputMode, source, send, holdCount, DAQdeviceNumber)

Purpose

Establishes the track/hold behavior of an SCXI-1140 module and sets up the module for either a single-channel operation or an interval-scanning operation.

Parameters

Input

Name	Type		Description
	Windows	Windows NT	
SCXIchassisID	I16	I32	logical ID assigned to the SCXI chassis
moduleSlot	I16	I32	chassis slot number
inputMode	I16	I32	type of analog input operation
source	I16	I32	indicates which signal will control the track/hold state
send	I16	I32	where else to send the signal specified by **source**
holdCount	I16	I32	number of times the module is enabled during an interval scan before going back into track mode
DAQdeviceNumber	I16	I32	device number of the DAQ device used

SCXI_Track_Hold_Setup

Continued

Parameter Discussion

inputMode indicates what type of analog input operation.

0: None; frees any resources that were previously reserved for the module (such as a DAQ device counter or an SCXIbus trigger line).

1: Single-channel operation.

2: Interval channel-scanning operation (only supported if the **DAQdeviceNumber** specified is an MIO or AI device, Lab-PC-1200, Lab-PC-1200AI, Lab-PC+, SCXI-1200, or DAQCard-1200).

source indicates what signal controls the track/hold state of the module. If the **inputMode** is 0, NI-DAQ ignores this parameter.

0: A counter of the DAQ device that is cabled to the module will be the source (NI-DAQ will reserve and use Am9513-based device counter 2, an E Series dedicated DAQ-STC counter, Lab-PC-1200, Lab-PC-1200AI, Lab-PC+, or DAQCard-1200 counter B1, DAQCard-700 or LPM device counter 2 for this purpose). This source is only valid if the module is cabled to a DAQ device.

1: An external signal connected to the HOLDTRIG pin on the front connector of the module will control the track/hold state of the module. There is a hardware connection between the HOLDTRIG pin and the counter output of the DAQ device, so if **source** = 1 the appropriate counter (listed above) is driven by the external signal and will be reserved. Note that if **inputMode** = 2, this external signal will drive the scan interval timer. If you are using a Lab-PC-1200, Lab-PC-1200AI, Lab-PC+, SCXI-1200, DAQCard-1200, DAQCard-700, or LPM device, you must change the jumper setting on the SCXI-1341 or SCXI-1342 adapter device to prevent the external signal from damaging the timer chip on the DAQ device.

2: NI-DAQ will use a signal routed on an SCXIbus trigger line from another SCXI-1140 module to control the track/hold state of the module. If you are using an SCXI-1200 to control the SCXI-1140, you must use this option to route the trigger signal from the SCXI-1200 on the backplane.

send indicates where else to send the signal specified by source for synchronization purposes. NI-DAQ also ignores this parameter if the **inputMode** is 0.

0: Nowhere.

1: Make the **source** signal drive the DAQ device counter output and the HOLDTRIG pin on the module front connector (if the **source** is not already one of those signals). If you are using a DAQCard-700, DAQCard-1200, Lab-PC-1200, Lab-PC-1200AI, Lab-PC+, or LPM device, you must change

SCXI_Track_Hold_Setup

Continued

the jumper setting on the SCXI-1341 or SCXI-1342 adapter device to prevent the external signal from damaging the timer chip on the DAQ device.

2:	Make the **source** signal drive an SCXIbus trigger line so that other SCXI-1140 modules can use it (if the **source** is not from the SCXIbus). Only one SCXI-1140 module can drive that trigger line; an error will occur if you attempt to configure more than one SCXI-1140 to drive it.

holdCount is the number of times the module is enabled by NI-DAQ during an interval scan before going back into track mode. Each time Slot 0 encounters an entry for the module in the module scan list, NI-DAQ enables the module, which remains enabled until the sample count in that module scan list entry expires. If there is only one entry for the module in the module scan list, **holdCount** should be 1 (this will almost always be the case).
Range: 1 to 255.

DAQdeviceNumber is the device number of the DAQ device in the PC that will be used to acquire the data. If the **DAQdeviceNumber** specified is a Lab-PC-1200, Lab-PC-1200AI, Lab-PC+, DAQCard-1200, DAQCard-700, or LPM device, **inputMode** 2 is not supported. If you are using the SCXI-1200 to acquire the data, use the logical device number you assigned to the SCXI-1200 in the configuration utility.

Using This Function

For single channel operations (**inputMode** = 1) the module is level-sensitive to the **source** signal; that is, when the **source** signal is low the module is in track mode, and when the **source** signal is high the module is in hold mode. If **source** = 0, you can use calls to `SCXI_Track_Hold_Control` function to put the module into track or hold mode by toggling the output of the appropriate counter on the DAQ device. If the SCXI-1140 you want to read is not cabled to the DAQ device, you will have to configure the SCXI-1140 module that *is* cabled to the DAQ device to send the counter output on the SCXIbus to the module you want. Then the `SCXI_Track_Hold_Control` call can put the module you want into track or hold mode. The `SCXI_Track_Hold_Setup` parameters for each module would be:

- For the SCXI-1140 that is cabled to the DAQ device as follows:
  inputMode = 1.
  source = 0.
  send = 2.
- For the SCXI-1140 module to be read:
  inputMode = 1.

SCXI_Track_Hold_Setup

Continued

source = 2.
send = 0.

Using an external **source** (**source** = 1) for single channel operations is not normally useful because NI-DAQ has no way of determining when the module has gone into hold mode and it is appropriate to read the channels.

(MIO, Lab-PC-1200, Lab-PC-1200AI, Lab-PC+, SCXI-1200, and DAQCard-1200 only) For interval channel scanning operations (**inputMode** = 2) NI-DAQ configures the module to go into hold mode on the rising edge of the **source** signal. If **source** = 0, that will happen when counter 2 on the Am9513-based devices, a dedicated DAQ-STC counter on E Series devices, or counter B1 on the Lab-PC-1200, Lab-PC-1200AI, Lab-PC+, SCXI-1200, or DAQCard-1200 pulses at the beginning of each scan interval; if **source** = 1, that will happen on the rising edge of the external signal connected to HOLDTRIG on the module front connector. In the latter case, you should configure the DAQ device for external scan interval timing (using the `DAQ_Config` function) so that the external signal will trigger each scan. If you want to scan more than one SCXI-1140, you can send the **source** signal from the module that is receiving it (either from the counter or from HOLDTRIG) to the other modules over the SCXIbus. Notice that the module that is cabled to the device can receive the **source** signal from the SCXIbus and drive the scan interval timer of the DAQ device, if you want; or the module can use the DAQ device counter output and send the signal on the SCXIbus, *even if that module is not in the module scan list.*

For example, you want to scan two SCXI-1140 modules; one of which is cabled to the DAQ device that is to perform the acquisition. An external signal connected to the HOLDTRIG pin of the module that is *not* cabled to the DAQ device is to control the track/hold state of both modules and the scan interval during the acquisition. The `SCXI_Track_Hold_Setup` parameters would be as follows:

- For the SCXI-1140 that is cabled to the DAQ device:
  inputMode = 2.
  source = 2.
  send = 1.
- For the other SCXI-1140 module to be scanned:
  inputMode = 2.
  source = 1.
  send = 2.

SCXI_Track_Hold_Setup

Continued

Remember to call the `DAQ_Config` function to enable external scan interval timing whenever the **source** signal of a module will be driving the scan interval counter, as in the previous example.

The module will go back into track mode after *n* module scan list entries for that module have occurred, where *n* is the **holdCount.** Usually, each module is represented in the module scan list only once, so a **holdCount** of one is appropriate. However, if an SCXI-1140 module is represented more than once in the module scan list and you want the module to remain in hold mode until after the last scan list entry for that module, you will need to set the module **holdCount** to equal the number of times the module is represented in the module scan list.

Select_Signal

Format

status = Select_Signal (deviceNumber, signal, source, sourceSpec)

Purpose

Chooses the source and polarity of a signal that the device uses (E Series devices only).

Note: ***NI-DAQ for Windows NT does not support this function.***

Parameters

Input

Name	Type		Description
	Windows	Windows NT	
deviceNumber	I16	n/a	assigned by configuration utility
signal	U32	n/a	signal for which you want to select the source and polarity
source	U32	n/a	the source of the signal
sourceSpec	U32	n/a	further signal specification (the polarity of the signal)

Parameter Discussion

Legal ranges for the **signal**, **source**, and **sourceSpec** parameters are given in terms of constants that are defined in a header file. The header file you should use depends on the language you are using:

- C programmers—`NIDAQCNS.H` (`DATAACQ.H` for LabWindows/CVI)
- BASIC programmers—`NIDAQCNS.INC` (Visual Basic for Windows programmers should refer to the *Programming Language Considerations* section in Chapter 1, *Using the NI-DAQ Functions*, for more information.)
- Pascal programmers—`NIDAQCNS.PAS`

Select_Signal

Continued

You can use the onboard DAQ-STC to select among many sources for various signals.

Use the **signal** parameter to specify the signal whose source you want to select. The following table shows the possible values for **signal**.

☞ **Note:** ***The* `ND_OUT_START_TRIGGER`*,* `ND_OUT_UPDATE`*, and* `ND_UPDATE_CLOCK_TIMEBASE` *values do not apply to the AI E Series devices.***

Group	signal	Description
Timing and Control Signals Used Internally by the Onboard DAQ-STC	`ND_IN_START_TRIGGER`	Start trigger for the `DAQ` and `SCAN` functions
	`ND_IN_STOP_TRIGGER`	Stop trigger for the `DAQ` and `SCAN` functions
	`ND_IN_SCAN_CLOCK_TIMEBASE`	Scan clock timebase for the `SCAN` functions
	`ND_IN_CHANNEL_CLOCK_TIMEBASE`	Channel clock timebase for the `DAQ` and `SCAN` functions
	`ND_IN_CONVERT`	Convert signal for the `AI`, `DAQ` and `SCAN` functions
	`ND_IN_SCAN_START`	Start scan signal for the `SCAN` functions
	`ND_IN_EXTERNAL_GATE`	External gate signal for the `DAQ` and `SCAN` functions
	`ND_OUT_START_TRIGGER`	Start trigger for the `WFM` functions
	`ND_OUT_UPDATE`	Update signal for the `AO` and `WFM` functions
	`ND_OUT_UPDATE_CLOCK_TIMEBASE`	Update clock timebase for the `WFM` functions

Select_Signal

Continued

Group	signal	Description
I/O Connector Pins	`ND_PFI_0` through `PFI_9`	Signal present at the I/O connector pin PFI0 through PFI9.
	`ND_GPCTR0_OUTPUT`	Signal present at the I/O connector pin GPCTR0_OUTPUT
	`ND_GPCTR1_OUTPUT`	Signal present at the I/O connector pin GPCTR1_OUTPUT
	`ND_FREQ_OUT`	Signal present at the FREQ_OUT output pin on the I/O connector.
RTSI Bus Signals	`ND_RTSI_0` through `ND_RTSI_6`	Signal present at the RTSI bus trigger line 0 through 7.
	`ND_RTSI_CLOCK`	Enable the device to drive the RTSI clock line or prevent it from doing it.
	`ND_BOARD_CLOCK`	Enable the device to receive the clock signal from the RTSI clock line or stop it from doing so.

Legal values for **source** and **sourceSpec** depend on the **signal** and are shown in the following tables:

signal = `ND_IN_START_TRIGGER`

source	sourceSpec
`ND_PFI_0` through `ND_PFI_9`	`ND_LOW_TO_HIGH` and `ND_HIGH_TO_LOW`
`ND_RTSI_0` through `ND_RTSI_6`	`ND_LOW_TO_HIGH` and `ND_HIGH_TO_LOW`
`ND_GPCTR0_OUTPUT`	`ND_LOW_TO_HIGH` and `ND_HIGH_TO_LOW`
`ND_AUTOMATIC`	`ND_DONT_CARE`

Select_Signal

Continued

Use ND_IN_START_TRIGGER to initiate a data acquisition sequence. You can use an external signal or output of general-purpose counter 0 as a source for this signal, or you can specify that NI-DAQ generates it (corresponds to **source** = ND_AUTOMATIC).

If you do not call this function with **signal** = ND_IN_START_TRIGGER, NI-DAQ uses the default values, **source** = ND_AUTOMATIC and **sourceSpec** = ND_LOW_TO_HIGH.

If you call DAQ_Config with **startTrig** = 1, NI-DAQ calls Select_Signal function with **signal** = ND_IN_START_TRIGGER, **source** = ND_PFI_0, and **sourceSpec** = ND_HIGH_TO_LOW.

If you call DAQ_Config with **startTrig** = 0, NI-DAQ calls Select_Signal function with **signal** = ND_IN_START_TRIGGER, **source** = ND_AUTOMATIC, and **sourceSpec** = ND_DONT_CARE.

signal = ND_IN_STOP_TRIGGER

source	**sourceSpec**
ND_PFI_0 through ND_PFI_9	ND_LOW_TO_HIGH and ND_HIGH_TO_LOW
ND_RTSI_0 through ND_RTSI_6	ND_LOW_TO_HIGH and ND_HIGH_TO_LOW

Use ND_IN_STOP_TRIGGER for data acquisition in the pretriggered mode. The selected transition on the **signal** line indicates to the device that it should acquire a specified number of scans after the trigger and stop.

If you do not call this function with **signal** = ND_IN_STOP_TRIGGER, NI-DAQ uses the default values, **source** = ND_PFI_1 and **sourceSpec** = ND_HIGH_TO_LOW. By default, ND_IN_STOP_TRIGGER is not used because the pretriggered mode is disabled.

If you call DAQ_StopTrigger_Config with **startTrig** = 1, NI-DAQ calls Select_Signal function with **signal** = ND_IN_STOP_TRIGGER, **source** = ND_PFI_1, and **sourceSpec** = ND_HIGH_TO_LOW. Therefore, if you want to use different selection for ND_IN_STOP_TRIGGER, you need to call the Select_Signal function after DAQ_StopTrigger_Config.

Select_Signal

Continued

signal = `ND_IN_EXTERNAL_GATE`

source	**sourceSpec**
`ND_PFI_0` through `ND_PFI_9`	`ND_PAUSE_ON_HIGH` and `ND_PAUSE_ON_LOW`
`ND_RTSI_0` through `ND_RTSI_6`	`ND_PAUSE_ON_HIGH` and `ND_PAUSE_ON_LOW`
`ND_NONE`	`ND_DONT_CARE`

Use `ND_IN_EXTERNAL_GATE` for gating the data acquisition. For example, if you call this function with **signal** = `ND_IN_EXTERNAL_GATE`, **source** = `ND_PFI_9`, and **sourceSpec** = `PAUSE_ON_HIGH`, the data acquisition will be paused whenever the PFI 9 is at the high level. The pausing is performed on a per scan basis, so no scans are split by the external gate.

If you do not call this function with **signal** = `ND_IN_EXTERNAL_GATE`, NI-DAQ uses the default values, **source** = `ND_NONE` and **sourceSpec** = `ND_DONT_CARE`; therefore, by default, the data acquisition is not gated.

signal = `ND_IN_SCAN_START`

source	**sourceSpec**
`ND_PFI_0` through `ND_PFI_9`	`ND_LOW_TO_HIGH` and `ND_HIGH_TO_LOW`
`ND_RTSI_0` through `ND_RTSI_6`	`ND_LOW_TO_HIGH` and `ND_HIGH_TO_LOW`
`ND_GPCTR0_OUTPUT`	`ND_LOW_TO_HIGH` and `ND_HIGH_TO_LOW`
`ND_INTERNAL_TIMER`	`ND_LOW_TO_HIGH`

Use this signal for scan timing. You can use a DAQ-STC timer for timing the scans, or you can use an external signal. You can also use the output of the general-purpose counter 0 for scan timing. This can be useful for applications such as Equivalent Time Sampling (ETS).

Select_Signal

Continued

If you do not call this function with **signal** = `ND_IN_SCAN_START`, NI-DAQ uses the default values, **source** = `ND_INTERNAL_TIMER` and **sourceSpec** = `ND_LOW_TO_HIGH`.

If you call `DAQ_Config` with **extConv** = 2 or 3, NI-DAQ calls `Select_Signal` function with **signal** = `ND_IN_SCAN_START`, **source** = `ND_PFI_7`, and **sourceSpec** = `ND_HIGH_TO_LOW`.

If you call `DAQ_Config` with **extConv** = 0 or 1, NI-DAQ calls `Select_Signal` function with **signal** = `ND_IN_SCAN_START`, **source** = `ND_INTERNAL_TIMER`, and **sourceSpec** = `ND_LOW_TO_HIGH`.

signal = `ND_IN_CONVERT`

source	**sourceSpec**
`ND_PFI_0` through `ND_PFI_9`	`ND_LOW_TO_HIGH` and `ND_HIGH_TO_LOW`
`ND_RTSI_0` through `ND_RTSI_6`	`ND_LOW_TO_HIGH` and `ND_HIGH_TO_LOW`
`ND_GPCTR0_OUTPUT`	`ND_LOW_TO_HIGH` and `ND_HIGH_TO_LOW`
`ND_INTERNAL_TIMER`	`ND_LOW_TO_HIGH`

Use `ND_IN_CONVERT` for sample (channel interval) timing. This signal controls the onboard ADC. You can use a DAQ-STC timer for timing the samples, or you can use an external signal. You can also use output of the general-purpose counter 0 for sample timing.

If you do not call this function with **signal** = `ND_IN_CONVERT`, NI-DAQ uses the default values, **source** = `ND_INTERNAL_TIMER` and **sourceSpec** = `ND_LOW_TO_HIGH`.

If you call `DAQ_Config` with **extConv** = 1 or 3, NI-DAQ calls `Select_Signal` function with **signal** = `ND_IN_CONVERT`, **source** = `ND_PFI_2`, and **sourceSpec** = `ND_HIGH_TO_LOW`.

If you call `DAQ_Config` with **extConv** = 0 or 2, NI-DAQ calls `Select_Signal` function with **signal** = `ND_IN_CONVERT`, **source** = `ND_INTERNAL_TIMER`, and **sourceSpec** = `ND_LOW_TO_HIGH`.

Select_Signal

Continued

signal = `ND_IN_SCAN_CLOCK_TIMEBASE`

source	**sourceSpec**
`ND_PFI_0` through `ND_PFI_9`	`ND_LOW_TO_HIGH` and `ND_HIGH_TO_LOW`
`ND_RTSI_0` through `ND_RTSI_6`	`ND_LOW_TO_HIGH` and `ND_HIGH_TO_LOW`
`ND_INTERNAL_20_MHZ`	`ND_LOW_TO_HIGH`
`ND_INTERNAL_100_KHZ`	`ND_LOW_TO_HIGH`

Use `ND_IN_SCAN_CLOCK_TIMEBASE` as an input into the DAQ-STC scan timer. The scan timer generates timing by counting the signal at its input, and producing an `IN_START_SCAN` signal after the specified number of occurrences of the `ND_IN_SCAN_CLOCK_TIMEBASE` signal transitions.

If you do not call this function with **signal** = `ND_IN_SCAN_CLOCK_TIMEBASE`, NI-DAQ uses the default values, **source** = `ND_INTERNAL_20_MHZ` and **sourceSpec** = `ND_LOW_TO_HIGH`.

signal = `ND_IN_CHANNEL_CLOCK_TIMEBASE`

source	**sourceSpec**
`ND_PFI_0` through `ND_PFI_9`	`ND_LOW_TO_HIGH` and `ND_HIGH_TO_LOW`
`ND_RTSI_0` through `ND_RTSI_6`	`ND_LOW_TO_HIGH` and `ND_HIGH_TO_LOW`
`ND_INTERNAL_20_MHZ`	`ND_LOW_TO_HIGH`
`ND_INTERNAL_100_KHZ`	`ND_LOW_TO_HIGH`

Use `ND_IN_CHANNEL_CLOCK_TIMEBASE` as an input into the DAQ-STC sample (channel interval) timer. The sample timer generates timing by counting the signal at its input, and producing an `ND_IN_CONVERT` signal after the specified number of occurrences of the `ND_IN_CHANNEL_CLOCK_TIMEBASE` signal transitions.

Select_Signal

Continued

If you do not call this function with **signal** = `ND_IN_SCAN_CLOCK_TIMEBASE`, NI-DAQ uses the default values, **source** = `ND_INTERNAL_20_MHZ` and **sourceSpec** = `ND_LOW_TO_HIGH`.

signal = `ND_OUT_START_TRIGGER`

source	**sourceSpec**
`ND_PFI_0` through `ND_PFI_9`	`ND_LOW_TO_HIGH` and `ND_HIGH_TO_LOW`
`ND_RTSI_0` through `ND_RTSI_6`	`ND_LOW_TO_HIGH` and `ND_HIGH_TO_LOW`
`ND_IN_START_TRIGGER`	`ND_LOW_TO_HIGH`
`ND_AUTOMATIC`	`ND_LOW_TO_HIGH`

Use `ND_OUT_START_TRIGGER` to initiate a waveform generation sequence. You can use an external signal or the signal used as the `ND_IN_START_TRIGGER`, or NI-DAQ can generate it. Setting source to `ND_IN_START_TRIGGER` is useful for synchronizing waveform generation with data acquisition.

If you do not call this function with **signal** = `ND_OUT_START_TRIGGER`, NI-DAQ uses the default values, **source** = `ND_AUTOMATIC` and **sourceSpec** = `ND_LOW_TO_HIGH`.

signal = `ND_OUT_UPDATE`

source	**sourceSpec**
`ND_PFI_0` through `ND_PFI_9`	`ND_LOW_TO_HIGH` and `ND_HIGH_TO_LOW`
`ND_RTSI_0` through `ND_RTSI_6`	`ND_LOW_TO_HIGH` and `ND_HIGH_TO_LOW`
`ND_GPCTR1_OUTPUT`	`ND_LOW_TO_HIGH` and `ND_HIGH_TO_LOW`
`ND_INTERNAL_TIMER`	`ND_LOW_TO_HIGH`

Use this signal for update timing. You can use a DAQ-STC timer for timing the updates, or you can use an external signal. You can also use output of the general-purpose counter 1 for update timing.

Select_Signal

Continued

If you do not call this function with **signal** = ND_OUT_UPDATE, NI-DAQ uses the default values, **source** = ND_INTERNAL_TIMER and **sourceSpec** = ND_LOW_TO_HIGH.

signal = ND_OUT_UPDATE_CLOCK_TIMEBASE

source	sourceSpec
ND_PFI_0 through ND_PFI_9	ND_LOW_TO_HIGH and ND_HIGH_TO_LOW
ND_RTSI_0 through ND_RTSI_6	ND_LOW_TO_HIGH and ND_HIGH_TO_LOW
ND_INTERNAL_20_MHZ	ND_LOW_TO_HIGH
ND_INTERNAL_100_KHZ	ND_LOW_TO_HIGH

Use this signal as an input into the DAQ-STC update timer. The update timer generates timing by counting the signal at its input and producing an ND_OUT_UPDATE signal after the specified number of occurrences of the ND_OUT_UPDATE_CLOCK_TIMEBASE signal transitions.

If you do not call this function with **signal** = ND_OUT_UPDATE_CLOCK_TIMEBASE, NI-DAQ uses the default values, **source** = ND_INTERNAL_20_MHZ and **sourceSpec** = ND_LOW_TO_HIGH.

signal = ND_PFI_0 **through** ND_PFI_9
The following table summarizes all the signals and source for the I/O connector pins PFI0 through PFI9.

signal	source	sourceSpec
ND_PFI_0 through ND_PFI_9	ND_NONE	ND_DONT_CARE
ND_PFI_0	ND_IN_START_TRIGGER	ND_LOW_TO_HIGH
ND_PFI_1	ND_IN_STOP_TRIGGER	ND_LOW_TO_HIGH
ND_PFI_2	ND_IN_CONVERT	ND_HIGH_TO_LOW

Select_Signal

Continued

signal	**source**	**sourceSpec**
ND_PFI_3	ND_GPCTR1_SOURCE	ND_LOW_TO_HIGH
ND_PFI_4	ND_GPCTR1_GATE	ND_POSITIVE
ND_PFI_5	ND_OUT_UPDATE	ND_HIGH_TO_LOW
ND_PFI_6	ND_OUT_START_TRIGGER	ND_LOW_TO_HIGH
ND_PFI_7	ND_IN_SCAN_START	ND_LOW_TO_HIGH
ND_PFI_7	ND_IN_SCAN_IH_PROG	ND_LOW_TO_HIGH
ND_PFI_8	ND_GPCTR0_SOURCE	ND_LOW_TO_HIGH
ND_PFI_9	ND_GPCTR0_GATE	ND_POSITIVE

Use ND_NONE to disable output on the pin.

signal = ND_GPCTR0_OUTPUT

source	**sourceSpec**
ND_NONE	ND_DONT_CARE
ND_GPCTR0_OUTPUT	ND_LOW_TO_HIGH
ND_RTSI_0 through ND_RTSI_6	ND_LOW_TO_HIGH

Use ND_NONE to disable output on the pin. When you disable output on this pin, you can use the pin as an input pin, and you can attach an external signal to it. This is useful because it enables you to communicate a signal from the I/O connector to the RTSI bus.

When you enable this pin for output, you can program it to output the signal present at any one of the RTSI bus trigger lines or the general-purpose counter 0 output. The RTSI selections are useful because they enable you to communicate a signal from the RTSI bus to the I/O connector.

Select_Signal

Continued

signal = `ND_GPCTR1_OUTPUT`

source	**sourceSpec**
`ND_NONE`	`ND_DONT_CARE`
`ND_GPCTR1_OUTPUT`	`ND_LOW_TO_HIGH`
`ND_RESERVED`	`ND_DONT_CARE`

Use `ND_NONE` to disable the output on the pin, in other words, do place the pin in high impedance state.

NI-DAQ may use `ND_RESERVED` when you use this device with some of the SCXI modules. In this case, you can use general-purpose counter 1, but the output will not be available on the I/O connector because the pin is used for device-to-SCXI communication. Currently, there are no SCXI modules that require this.

signal = `ND_FREQ_OUT`

source	**sourceSpec**
`ND_NONE`	`ND_DONT_CARE`
`ND_INTERNAL_10_MHZ`	1 through 16
`ND_INTERNAL_100_KHZ`	1 through 16

Use `ND_NONE` to disable the output on the pin.

The signal present on the FREQ_OUT pin of the I/O connector is the divided-down version of one of the two internal timebases. Use **sourceSpec** to specify the divide-down factor.

signal = `ND_RTSI_0` **through** `ND_RTSI_6`

source	**sourceSpec**
`ND_NONE`	`ND_DONT_CARE`

Select_Signal

Continued

source	**sourceSpec**
ND_IN_START_TRIGGER	ND_LOW_TO_HIGH
ND_IN_STOP_TRIGGER	ND_LOW_TO_HIGH
ND_IN_CONVERT	ND_HIGH_TO_LOW
ND_OUT_UPDATE	ND_HIGH_TO_LOW
ND_OUT_START_TRIGGER	ND_LOW_TO_HIGH
ND_GPCTR0_SOURCE	ND_LOW_TO_HIGH
ND_GPCTR0_GATE	ND_POSITIVE
ND_GPCTR0_OUTPUT	ND_DONT_CARE

Use `ND_NONE` to disable output on the RTSI line.

You can use the GPCTR0_OUTPUT pin on the I/O connector in two ways—as an output pin or an input pin. When you configure the pin as an output pin, you can program the pin to output a signal from a RTSI line or the general-purpose counter 0 output (see **signal** = `ND_GPCTR0_OUTPUT` in this function for details). When you configure the pin as an input pin, you can attach an external signal to the pin. When **signal** is one of the RTSI lines, and **source** = `ND_GPCTR0_OUTPUT`, the signal on the RTSI line will be the signal present at the GPCTR0_OUTPUT pin on the I/O connector, which is not always the output of the general-purpose counter 0.

signal = `ND_RTSI_CLOCK`

source	**sourceSpec**
ND_NONE	ND_DONT_CARE
ND_BOARD_CLOCK	ND_DONT_CARE

Use **source** = `ND_NONE` to stop the device from driving the RTSI clock line.

Select_Signal

Continued

When **source** = `ND_BOARD_CLOCK`, this device drives the signal on the RTSI clock line.

signal = `ND_BOARD_CLOCK`

source	sourceSpec
`ND_BOARD_CLOCK`	`ND_DONT_CARE`
`ND_RTSI_CLOCK`	`ND_DONT_CARE`

Use **source** = `ND_BOARD_CLOCK` to stop the device from receiving the clock signal from the RTSI clock line.

Use **source** = `ND_RTSI_CLOCK` to program the device to receive the clock signal from the RTSI clock line.

Using This Function

If you have selected a signal that is not an I/O connector pin or a RTSI bus line, `Select_Signal` saves the parameters in the configuration tables for future operations. Functions which initiate data acquisition (`DAQ_Start`, `SCAN_Start`, `DAQ_Op`, and `SCAN_Op`) and waveform generation operations (`WFM_Group_Control` and `WFM_Op`) use the configuration tables to set the device circuitry to the correct timing modes.

You do not need to call this function if you are satisfied with the default settings for the signals.

If you have selected a signal that is an I/O connector or a RTSI bus signal, `Select_Signal` performs signal routing and enables or disables output on a pin or a RTSI line.

Example: Sending a signal from your E Series device to the RTSI bus

To send a signal from your E Series device to the RTSI bus, set **signal** to the appropriate RTSI bus line and **source** to indicate the signal from your device. If you want to send the analog input start trigger on to RTSI line 3, use the following call:

```
Select_Signal(deviceNum, ND_RTSI_3, ND_IN_START_TRIGGER, ND_LOW_TO_HIGH)
```

Select_Signal

Continued

Example: Receiving a signal from the RTSI bus on your E Series device

To receive a signal from the RTSI bus and use it as a signal on your E Series device, set **signal** to indicate the appropriate E Series device signal and **source** to the appropriate RTSI line. If you want to use low-to-high transitions of the signal present on the RTSI line 4 as your scan clock, use the following call:

```
Select_Signal(deviceNum, ND_IN_SCAN_START, ND_RTSI_4, ND_LOW_TO_HIGH)
```

Signal Name Equivalencies: For a variety of reasons, some timing signals are given different names in the hardware documentation and the software and its documentation. The following table lists the equivalencies between the two sets of signal names.

Table 2-9. E Series Signal Name Equivalencies

	Hardware Name	**Software Name**
AI-Related Signals	TRIG1	`ND_IN_START_TRIGGER`
	TRIG2	`ND_IN_STOP_TRIGGER`
	STARTSCAN	`ND_IN_SCAN_START`
	SISOURCE	`ND_IN_SCAN_CLOCK_TIMEBASE`
	CONVERT*	`ND_IN_CONVERT`
	AIGATE	`ND_IN_EXTERNAL_GATE`
	SI2SOURCE	`ND_IN_CHANNEL_CLOCK_TIMEBASE`
AO-Related Signals	WFTRIG	`ND_OUT_START_TRIGGER`
	UPDATE*	`ND_OUT_UPDATE`
	AOGATE	`ND_OUT_EXTERNAL_GATE`
	UISOURCE	`ND_OUT_UPDATE_CLOCK_TIMEBASE`
	AO2GATE	
	UI2SOURCE	

Set_DAQ_Device_Info

Format

status = Set_DAQ_Device_Info (deviceNumber, infoType, infoValue)

Purpose

This function can be used to change the data transfer mode (interrupts and DMA) for certain classes of data acquisition operations, some settings for an SC-2040 track-and-hold accessory and an SC-2043-SG strain-gauge accessory, as well as the source for the CLK1 signal on the DAQCard-700.

Parameters

Input

Name	Type		Description
	Windows	Windows NT	
deviceNumber	I16	I32	assigned by configuration utility
infoType	U32	U32	parameter you want to modify
infoValue	U32	U32	new value you want to assign to the parameter specified by **infoType**

Parameter Discussion

Legal ranges for the **infoType** and **infoValue** are given in terms of constants that are defined in a header file. The header file you should use depends on the language you are using:

- C programmers—`NIDAQCNS.H` (`DATAACQ.H` for LabWindows/CVI)
- BASIC programmers—`NIDAQCNS.INC` (Visual Basic for Windows programmers should refer to the *Programming Language Considerations* section in Chapter 1, *Using the NI-DAQ Functions*, for more information.)
- Pascal programmers—`NIDAQCNS.PAS`

Set_DAQ_Device_Info

Continued

Use **infoType** to let NI-DAQ know which parameter you want to change. Use **infoValue** to specify the corresponding new value.

Values that **infoType** accepts depend on the device you are using. The legal range for **infoValue** depends on the device you are using and **infoType**.

infoType can be one of the following:

infoType	Description
`ND_DATA_XFER_MODE_AI`	Method NI-DAQ will use for data transfers when performing the `DAQ`, `MDAQ`, and `SCAN` operations.
`ND_DATA_XFER_MODE_AO_GR1` `ND_DATA_XFER_MODE_AO_GR2`	Method NI-DAQ will use for data transfers when performing the `WFM` operations which require buffers from the PC memory.
`ND_DATA_XFER_MODE_GPCTR0`	Method NI-DAQ will use for data transfers when buffered `GPCTR` operations with the general purpose counter 0.
`ND_DATA_XFER_MODE_GPCTR1`	Method NI-DAQ will use for data transfers when buffered GPCTR operations with the general-purpose counter 1.
`ND_DATA_XFER_MODE_DIO_GR1`	Method NI-DAQ will use for data transfers for the digital input and output operations with group 1.
`ND_DATA_XFER_MODE_DIO_GR2`	Method NI-DAQ will use for data transfers for the digital input and output operations with group 2.
`ND_SC_2040_MODE`	Used to enable or disable the track-and-hold circuitry on the SC-2040.
`ND_SC_2043_MODE`	Used to enable or disable the SC-2043-SG accessory.
`ND_COUNTER_1_SOURCE`	Used to select a source for counter 1 on the DAQCard-700.

Set_DAQ_Device_Info

Continued

infoValue can be one of the following:

infoValue	Description
`ND_INTERRUPTS`	NI-DAQ will use interrupts for data transfers.
`ND_UP_TO_1_DMA_CHANNEL`	NI-DAQ will use one DMA channel, if possible; if the DMA channel is not available, NI-DAQ will report an error and it will not perform the operation.
`ND_UP_TO_2_DMA_CHANNELS`	NI-DAQ will use two DMA channels, if possible; otherwise, it will use one DMA channel, if one is available; if no DMA channels are available, NI-DAQ will report an error and it will not perform the operation.
`ND_NO_TRACK_AND_HOLD`	Disables use of the track-and-hold circuitry on the SC-2040.[1]
`ND_TRACK_AND_HOLD`	Re-enables the track-and-hold circuitry on an SC-2040 if you have previously disabled it.[2]
`ND_NONE`	Cancels the effects of having accidentally called the `SC_2040_Config` function.
`ND_STRAIN_GAUGE`	Enables the SC-2043-SG accessory for strain-gauge measurements (no excitation on channel 0).
`ND_STRAIN_GAUGE_EX0`	Enables the SC-2043-SG accessory with excitation on channel 0.
`ND_NO_STRAIN_GAUGE`	Disables the SC-2043-SG accessory.
`ND_INTERNAL_TIMER`	Counter 1 will use the internal timer as the source for its CLK1 source.

Set_DAQ_Device_Info

Continued

infoValue	Description
ND_IO_CONNECTOR	Counter 1 will use the CLK1 signal from the I/O connector as the source for its CLK1 signal.

[1]You should use this setting if you want to use the SC-2040 only as a preamplifier, without using track and hold.
[2]with ND_NO_TRACK_AND_HOLD.

When NI-DAQ uses DMA channels for data transfers, it must have an interrupt level available for the device performing the transfers. In this case, NI-DAQ uses interrupts for DMA controller reprogramming and exception handling.

Using This Function

You can use this function to select the data transfer method for a given operation on a particular device. If you do not use this function, NI-DAQ will decide on the data transfer method that will typically take maximum advantage of available resources.

All possible data transfer methods for the devices supported by NI-DAQ are listed below. If your device is not listed, none of the data transfer modes are applicable. An asterisk is placed next to the default data transfer mode for each device.

Device Type	infoType	infoValue
AT-A2150	ND_DATA_XFER_MODE_AI	ND_INTERRUPTS ND_UP_TO_1_DMA_CHANNEL*
AT-AO-6/10	ND_DATA_XFER_MODE_AO_GR1	ND_INTERRUPTS ND_UP_TO_1_DMA_CHANNEL
AT-DIO-32F	ND_DATA_XFER_MODE_DIO_GR1	ND_UP_TO_1_DMA_CHANNEL ND_UP_TO_2_DMA_CHANNELS*
	ND_DATA_XFER_MODE_DIO_GR2	ND_UP_TO_1_DMA_CHANNEL*

Set_DAQ_Device_Info

Continued

Device Type	infoType	infoValue
AT-DSP2200	ND_DATA_XFER_MODE_AI	ND_INTERRUPTS* ND_UP_TO_1_DMA_CHANNEL
	ND_DATA_XFER_MODE_AO_GR1	ND_INTERRUPTS* ND_UP_TO_1_DMA_CHANNEL
AT-MIO-16 AT-MIO-16D	ND_DATA_XFER_MODE_AI	ND_INTERRUPTS ND_UP_TO_1_DMA_CHANNEL ND_UP_TO_2_DMA_CHANNELS*
	ND_DATA_XFER_MODE_AO	ND_INTERRUPTS*
AT-MIO-16E-1	ND_DATA_XFER_MODE_AI	ND_INTERRUPTS ND_UP_TO_1_DMA_CHANNEL* ND_UP_TO_2_DMA_CHANNELS
	ND_DATA_XFER_MODE_AO_GR1	ND_INTERRUPTS ND_UP_TO_1_DMA_CHANNEL*
	ND_DATA_XFER_MODE_GPCTR0	ND_INTERRUPTS ND_UP_TO_1_DMA_CHANNEL ND_UP_TO_2_DMA_CHANNELS*
	ND_DATA_XFER_MODE_GPCTR1	ND_INTERRUPTS ND_UP_TO_1_DMA_CHANNEL ND_UP_TO_2_DMA_CHANNELS*
AT-MIO-16E-2 NEC-MIO-16E-4 AT-MIO-64E-3	ND_DATA_XFER_MODE_AI	ND_INTERRUPTS ND_UP_TO_1_DMA_CHANNEL*
	ND_DATA_XFER_MODE_AO_GR1	ND_INTERRUPTS ND_UP_TO_1_DMA_CHANNEL*
	ND_DATA_XFER_MODE_GPCTR0	ND_INTERRUPTS ND_UP_TO_1_DMA_CHANNEL ND_UP_TO_2_DMA_CHANNELS*
	ND_DATA_XFER_MODE_GPCTR1	ND_INTERRUPTS ND_UP_TO_1_DMA_CHANNEL ND_UP_TO_2_DMA_CHANNELS*

Set_DAQ_Device_Info

Continued

Device Type	infoType	infoValue
AT-AI-16XE-10 NEC-AI-16E-4 NEC-AI-16XE-50	`ND_DATA_XFER_MODE_AI`	`ND_INTERRUPTS` `ND_UP_TO_1_DMA_CHANNEL*`

	`ND_DATA_XFER_MODE_GPCTR0`	`ND_INTERRUPTS`
		`ND_UP_TO_1_DMA_CHANNEL`
		`ND_UP_TO_2_DMA_CHANNELS*`

	`ND_DATA_XFER_MODE_GPCTR1`	`ND_INTERRUPTS`
		`ND_UP_TO_1_DMA_CHANNEL`
		`ND_UP_TO_2_DMA_CHANNELS*`

AT-MIO-16F-5
AT-MIO-16X
AT-MIO-64F-5
	`ND_DATA_XFER_MODE_AI`	`ND_INTERRUPTS`
		`ND_UP_TO_1_DMA_CHANNEL*`
		`ND_UP_TO_2_DMA_CHANNELS`

	`ND_DATA_XFER_MODE_AO_GR1`	`ND_INTERRUPTS`
		`ND_UP_TO_1_DMA_CHANNEL*`
		`ND_UP_TO_2_DMA_CHANNELS`

AT-MIO-16E-10
AT-MIO-16DE-10
AT-MIO-16XE-10
AT-MIO-16XE-50
NEC-MIO-16XE-50
	`ND_DATA_XFER_MODE_AI`	`ND_INTERRUPTS`
		`ND_UP_TO_1_DMA_CHANNEL`
		`ND_UP_TO_2_DMA_CHANNELS*`

	`ND_DATA_XFER_MODE_AO_GR1`	`ND_INTERRUPTS`
		`ND_UP_TO_1_DMA_CHANNEL*`
		`ND_UP_TO_2_DMA_CHANNELS`

	`ND_DATA_XFER_MODE_GPCTR0`	`ND_INTERRUPTS`
		`ND_UP_TO_1_DMA_CHANNEL`
		`ND_UP_TO_2_DMA_CHANNELS*`

	`ND_DATA_XFER_MODE_GPCTR1`	`ND_INTERRUPTS`
		`ND_UP_TO_1_DMA_CHANNEL`
		`ND_UP_TO_2_DMA_CHANNELS*`

DAQCard-AI-16E-4 DAQCard-AI-16XE-50	`ND_DATA_XFER_MODE_AI`	`ND_INTERRUPTS`
	`ND_DATA_XFER_MODE_GPCTR0`	`ND_INTERRUPTS`
	`ND_DATA_XFER_MODE_GPCTR1`	`ND_INTERRUPTS`

Set_DAQ_Device_Info

Continued

Device Type	infoType	infoValue
DAQPad-MIO-16XE-50	`ND_DATA_XFER_MODE_AI`	`ND_INTERRUPTS`
	`ND_DATA_XFER_MODE_AO_GR1`	`ND_INTERRUPTS*`
	`ND_DATA_XFER_MODE_GPCTR0`	`ND_INTERRUPTS`
	`ND_DATA_XFER_MODE_GPCTR1`	`ND_INTERRUPTS`
516 devices DAQCard-500/700	`ND_DATA_XFER_MODE_AI`	`ND_INTERRUPTS*`
EISA-A2000	`ND_DATA_XFER_MODE_AI`	`ND_INTERRUPTS` `ND_UP_TO_1_DMA_CHANNEL*`
Lab-PC+	`ND_DATA_XFER_MODE_AI`	`ND_INTERRUPTS` `ND_UP_TO_1_DMA_CHANNEL*`
LPM devices	`ND_DATA_XFER_MODE_AI`	`ND_INTERRUPTS*`
DAQCard-1200 DAQPad-1200 Lab-PC-1200 Lab-PC-1200AI SCXI-1200	`ND_DATA_XFER_MODE_AI`	`ND_INTERRUPTS*`
	`ND_DATA_XFER_MODE_AO_GR1`	`ND_INTERRUPTS*`

NI-DAQ uses interrupts and DMA channels for data transfers. The DMA data transfers are typically faster, so you may want to take advantage of them. Note that the data transfer modes `ND_UP_TO_1_DMA_CHANNEL` and `ND_UP_TO_2_DMA_CHANNELS` do not reserve the DMA channel or channels for a particular operation; they just authorize NI-DAQ to use them, if they are available.

(AT-MIO-16, AT-MIO-16D, AT-MIO-16F-5, AT-MIO-16X, AT-MIO-64F-5 only) If you are performing high-speed analog input, you may increase your performance by setting `ND_DATA_XFER_MODE_AI` to `ND_UP_TO_2_DMA_CHANNELS`. Using two DMA channels, these devices are able to chain across buffer boundaries caused by page breaks on AT-compatible computers or by buffer fragmentation caused by mapping virtual into physical memory. Notice that EISA computers provide their own DMA

Set_DAQ_Device_Info

Continued

chaining mechanism and a single DMA channel is all that is necessary on these machines.

(AT-MIO-16F-5, AT-MIO-16X, AT-MIO-64F-5 only) If you want to use separate DMA channels for each of the analog output channels, you have to set `ND_DATA_XFER_MODE_AO` to `ND_UP_TO_2_DMA_CHANNELS` and `ND_DATA_XFER_MODE_AI` to `ND_INTERRUPTS`.

(AT-DIO-32F only) If you are performing high-speed digital input or output for group 1, setting `ND_DATA_XFER_MODE_DIO_GR1` to `ND_UP_TO_2_DMA_CHANNELS` makes both DMA channels available and can increase your performance.

(AT-DSP2200 only) This device can support either one DMA channel or one interrupt level at any one time.

Timeout_Config

Format

status = Timeout_Config (deviceNumber, timeout)

Purpose

Establishes a timeout limit that is used by the synchronous functions to ensure that these functions will eventually return control to your application. Examples of synchronous functions are `DAQ_Op`, `DAQ_DB_Transfer`, and `WFM_from_Disk`.

Parameters

Input

Name	Type		Description
	Windows	Windows NT	
deviceNumber	I16	I32	assigned by configuration utility
timeout	I32	I32	number of timer ticks

Parameter Discussion

timeout is the number of timer ticks. The duration of a tick is 55 ms (0.055 s), and there are approximately 18 ticks/s.

-1:	Wait indefinitely (timeout disabled).
0 to 2^{31} - 1:	Wait **timeout** $*$ 0.055 s before returning.

Using This Function

The synchronous functions do not return control to your application until they have accomplished their task. If you have indicated a large amount of data and/or a slow acquisition or generation rate, you may want to terminate the function prematurely, short of restarting your computer. By calling `Timeout_Config` before calling the synchronous function, you can set an upper bound on the amount of time the synchronous function will take before returning. If the synchronous function returns the error code **timeOutError**, you know that the number of ticks indicated in the **timeout** parameter have elapsed and the synchronous function has returned because of the timeout.

Timeout_Config

Continued

The following is a list of the synchronous functions:

```
DAQ_DB_StrTransfer   DIG_DB_Transfer     WFM_DB_StrTransfer
DAQ_DB_Transfer      Lab_ISCAN_Op        WFM_DB_Transfer
DAQ_Op               Lab_ISCAN_to_Disk   WFM_from_Disk
DAQ_to_Disk          SCAN_Op             WFM_Op
DIG_DB_StrTransfer   SCAN_to_Disk
```

Trigger_Window_Config

Format

status = Trigger_Window_Config (deviceNumber, mode, windowSize)

Purpose

Configures the hysteresis analog trigger feature of an acquisition device (AT-A2150 and AT-DSP2200 only).

Parameters

Input

Name	Type		Description
	Windows	Windows NT	
deviceNumber	I16	I32	assigned by configuration utility
mode	I16	I32	whether to use a hysteresis window
windowSize	U16	U32	number of digitizing levels below or above the analog trigger level before the trigger level is recognized

Parameter Discussion

mode specifies whether to use a hysteresis window.

0:	Disable hysteresis window.
1:	Enable hysteresis window.

windowSize is the number of digitizing levels below the analog trigger level (for positive slope) or above the analog trigger level (for negative slope) that the input signal must go before a valid trigger crossing at the analog trigger level is recognized.

Range: 0 to 65,535 on the AT-A2150. Notice the following restrictions:

- When detecting a positive slope, trigger level - **windowSize** ≥ -32,768.
- When detecting a negative slope, trigger level + **windowSize** ≤ 32,767.

Trigger_Window_Config

Continued

Using This Function

The AT-A2150 provides hysteresis when looking for an analog trigger. Hysteresis acts like a noise filter, ensuring that NI-DAQ does not mistake small variations in the input signal for actual triggers. Select the amount of variation that NI-DAQ is to ignore by **windowSize**. For example, if you have selected positive slope, analog trigger level is 2,048, and **windowSize** is 256, the triggering scheme first looks for a sample that is at 1,792 or below—that is, 256 or more below the analog trigger level. After finding this sample, the triggering scheme starts looking for a sample that is at 2,048 (the analog trigger level) or above, and when the triggering scheme finds this sample, it triggers. If you have selected a negative slope, the triggering scheme first looks for a sample that is **windowSize** digitizing levels about the trigger level and then for a sample that is at or below the trigger level.

On the AT-A2150, analog triggering is implemented in hardware and NI-DAQ uses only the upper eight bits of the binary value of the trigger level and the upper eight bits of the hexadecimal value obtained by adding and subtracting **windowSize** to or from the trigger level, for triggering. For example, NI-DAQ treats both values 2,048 (800 hex) and 2,303 (8FF hex) for the trigger level as 2,048 on the AT-A2150.

WFM_Chan_Control

Format

status = WFM_Chan_Control (deviceNumber, chan, operation)

Purpose

Temporarily halts or restarts waveform generation for a single analog output channel.

☞ **Note:** ***This function is not supported for the MIO E Series devices.***

Parameters

Input

Name	Type		Description
	Windows	Windows NT	
deviceNumber	I16	I32	assigned by configuration utility
chan	I16	I32	analog output channel
operation	I16	I32	pause or resume

Parameter Discussion

chan is the analog output channel to be paused or restarted.

Range:	0 or 1 for most devices.
	0 through 5 for AT-AO-6.
	0 through 9 for AT-AO-10.

operation selects the operation to be performed on the output channel.

2 (PAUSE):	Temporarily halts waveform generation for the output channel. NI-DAQ maintains the last voltage written to the DAC indefinitely.
4 (RESUME):	Restarts waveform generation for the output channel previously halted by **operation** = PAUSE.

WFM_Chan_Control

Continued

Using This Function

When a waveform generation has been halted by executing PAUSE, the RESUME operation restarts the waveform exactly at the point in your buffer where it left off.

(AT-AO-6/10, AT-MIO-16X, and AT-MIO-64F-5 only) You can use the PAUSE and RESUME operations on group 1 output channels only if at least one of the following conditions is true:

- Group 1 consists of a single output channel.
- Group 1 is using interrupts instead of DMA.

(AT-AO-6/10, AT-MIO-16X, and AT-MIO-64F-5 only) You will see a FIFO lag effect when you pause or resume group 1 channels. When you execute PAUSE for a group 1 channel, the effective pause will not occur until the FIFO has finished writing all of the data remaining in the FIFO for the specified channel. The same is true for the RESUME operation on a group 1 channel. NI-DAQ cannot place data for the specified channel into the FIFO until the FIFO has emptied. Refer to the *FIFO Lag Effect on the MIO E Series, AT-AO-6/10, AT-MIO-16X, and AT-MIO-64F-5* section of Chapter 3, *Software Overview*, of the *NI-DAQ User Manual for PC Compatibles* for a more detailed discussion.

WFM_Check

Format

status = WFM_Check (deviceNumber, chan, wfmStopped, itersDone, pointsDone)

Purpose

Returns status information concerning a waveform generation operation.

Parameters

Input

Name	Type		Description
	Windows	Windows NT	
deviceNumber	I16	I32	assigned by configuration utility
chan	I16	I32	number of the analog output channel

Output

Name	Type		Description
	Windows	Windows NT	
wfmStopped	I16	I32	whether the waveform is still in progress
itersDone	U32	U32	number of buffer iterations completed
pointsDone	U32	U32	number of points written for the current buffer iteration

WFM_Check

Continued

Parameter Discussion

chan is the number of the analog output channel performing the waveform generation operation.

Range:	0 or 1 for most devices.
	0 through 5 for AT-AO-6.
	0 through 9 for AT-AO-10.

wfmStopped is a flag whose value indicates whether the waveform generation operation is still in progress. If the number of iterations indicated in the last `WFM_Load` call is 0, status is always 0.

0:	Ongoing operation.
1:	Complete operation.

itersDone returns the number of buffer iterations that have been completed.

pointsDone returns the number of points written to the analog output channels specified in **chan** for the current buffer iteration. For devices that have analog output FIFOs, **pointsDone** returns the number of points written to the FIFO if **chan** belongs to group 1. Refer to the following *Using This Function* section for more information.
Range: 0 to **count** - 1 where **count** is the parameter used in the last `WFM_Load` call.

☞ **Note:** ***C Programmers*—wfmStopped, itersDone, *and* pointsDone *are pass-by-reference parameters.***

Using This Function

`WFM_Check` returns status information concerning the progress of a waveform generation operation. It is useful in determining when an operation has completed and when you can initiate a new operation.

`WFM_Check` may return an incorrect **underFlowError** after a waveform has finished if NI-DAQ loaded the waveform with an iterations parameter that is greater than zero. This occurs when the update interval is relatively small compared to the speed of your computer. For example, using an AT-MIO-16F-5 and DMA on a 386 20 MHz AT machine with a CI of 23.0, `WFM_Check` returned an incorrect **underFlowError** when a waveform with an update interval of 10 µs or less finished with its iterations. In general, if a continuous waveform is error free at a given update interval on your computer, a terminating waveform will also be error free. So, you should be able to ignore a **underFlowError** returned by `WFM_Check` when the function returns the **wfmStopped** parameter with a value of 1.

WFM_Check

Continued

(AT-AO-6/10, AT-MIO-16X, MIO E Series devices, and AT-MIO-64F-5 only) A FIFO lag effect is seen for group 1 channels. **pointsDone** and **itersDone** indicate the number of buffer points currently written to the FIFO. There is a time lag from the point when the data is written to the FIFO to when the data is output to the DACs. This time lag is dependent upon the update rate. For example, if you had a buffer of 50 points that you wanted to send to analog output channel 0, the first call to `WFM_Check` would have **itersdone** = 20. The FIFO would be filled up with 20 cycles of your 50-point buffer. Refer to the *FIFO Lag Effect on the MIO E Series, AT-AO-6/10, AT-MIO-16X, and AT-MIO-64F-5* section of Chapter 3, *Software Overview*, of the *NI-DAQ User Manual for PC Compatibles* for a more detailed discussion. **wfmStopped** is also affected by the FIFO lag, since **wfmStopped** indicates when the last point is written to the FIFO.

☞ **Note:** ***(AT-MIO-16X, MIO E Series devices, AT-MIO-64F-5, and AT-AO-6/10 only) If you use FIFO mode waveform generation,* pointsDone *is always 0. If the generation is continuous (including pulsed waveform generation), the parameters* wfmStopped *and* itersDone *are always 0; otherwise* wfmStopped *and* itersDone *indicate the status of waveform generation operation.***

WFM_ClockRate

Format

status = WFM_ClockRate (deviceNumber, group, whichclock, timebase, interval, mode)

Purpose

Sets an update rate and a delay rate for a group of analog output channels.

Parameters

Input

Name	Type		Description
	Windows	Windows NT	
deviceNumber	I16	I32	assigned by configuration utility
group	I16	I32	group of analog output channels
whichclock	I16	I32	the update or delay clock
timebase	I16	I32	resolution
interval	U32	U32	timebase divisor
mode	I16	I32	enables the delay clock

Parameter Discussion

group is the group of analog output channels (see `WFM_Group_Setup`).

Range: 1 for most devices.
1 or 2 for the AT-AO-6/10.

whichclock indicates the type of clock:

0: The update clock (default).
1: The delay clock.
2: The delay clock prescalar 1 (MIO E Series devices only).
3: The delay clock prescalar 2 (MIO E Series devices only).

WFM_ClockRate

Continued

Notice that you can program the delay clock only on the AT-MIO-16X, AT-MIO-64F-5, and MIO E Series devices.

timebase is the timebase, or resolution, to be used in determining **interval**. **timebase** has the following possible values:

-3:	20 MHz clock used as a timebase (50 ns) (MIO E Series only).
-1:	5 MHz clock used as timebase (200 ns resolution) (AT-MIO-16F-5, AT-MIO-64F-5, and AT-MIO-16X only)
0:	If **whichclock** is equal to 0, the external clock is connected to OUT2 on the MIO-16 and AT-MIO-16D; to EXTDACUPDATE* on the AT-MIO-16F-5, AT-MIO-64F-5, and AT-MIO-16X; to EXTUPDATE on the AT-AO-6/10 and Lab and 1200 Series analog output devices, or to a pin chosen through the `Select_Signal` function on an MIO E Series device (default is PFI5). If **whichclock** is equal to 1, the external clock is connected to OUT2 on the AT-MIO-16X and AT-MIO-64F-5.
1:	1 MHz clock used as timebase (1 µs resolution).
2:	100 kHz clock used as timebase (10 µs resolution).
3:	10 kHz clock used as timebase (100 µs resolution).
4:	1 kHz clock used as timebase (1 ms resolution).
5:	100 Hz clock used as timebase (10 ms resolution).
6:	SOURCE1 used as timebase (Am9513-based devices only).
7:	SOURCE2 used as timebase (Am9513-based devices only).
8:	SOURCE3 used as timebase (Am9513-based devices only).
9:	SOURCE4 used as timebase (Am9513-based devices only).
10:	SOURCE5 used as timebase (Am9513-based devices only).
11:	External timebase (MIO E Series devices only). Connect your external timebase to PFI5, by default, or use the `Select_Signal` function to specify a different source.

For an AT-DSP2200, **timebase** has these values:

0:	51.2 kHz.
1:	48.0 kHz.
2:	44.1 kHz.
3:	32.0 kHz.

On the MIO-16 and AT-MIO-16D, **timebase** = 0 sets counter 2 to the high-impedance state, allowing its output level to be externally driven by a signal connected to the OUT2 pin on the I/O connector. On the Lab and 1200 Series analog output devices, **timebase** = 0 allows the signal applied to the EXTUPDATE pin on the I/O connector to control the DAC update. On the AT-AO-6/10, **timebase** = 0 allows the signal applied to the

WFM_ClockRate

Continued

EXTDACUPDATE from the I/O connector or RTSI bus to control the DAC update. On the AT-MIO-16F-5, AT-MIO-64F-5, and AT-MIO-16X, **timebase** = 0 allows the signal applied to the EXTDACUPDATE pin on the I/O connector to control the DAC update. Whenever an active low pulse is detected on one of these pins, the DAC in the group are updated. When **timebase** = 0, the value of **interval** is irrelevant. **timebase** = 1 through 5 selects one of the five available internal clock signals to be used in determining the update interval.

interval indicates the number of timebase units. If **whichclock** is 0, **interval** indicates the number of **timebase** units of time that elapse between voltage updates at the analog output channels in the group. If **whichclock** is 1, **interval** indicates the number of timebase units of time that elapse after reaching the last point in DAC FIFO before the next cycle begins. If **whichclock** is 2, interval indicates delay interval prescalar 1. If **whichclock** is 3, interval indicates delay interval prescalar 2.

Range:	2 through 65,535 for the MIO devices except for MIO E Series devices.
	2 through 16,777,216 for MIO E Series devices.
	1, 2, 4, or 8 for an AT-DSP2200.

The only internal timebases available on the MIO E Series devices are 20 MHz and 100 kHz. If you use a **timebase** other than -3 or 2 for these devices, NI-DAQ performs the appropriate scaling, if possible.

☞ **Note:** ***If you are using an SCXI-1200 with remote SCXI, the maximum rate will depend on the baud rate setting and*** **updateRate*****. Refer to the*** *SCXI-1200 User Manual* ***for more details.***

mode depends on the **whichclock** parameter.

Range:	0, 1, or 2 for MIO E Series devices.
	0 or 1 for the AT-MIO-16X and AT-MIO-64F-5.
	0 for all other devices.

whichclock = 0:
When **whichclock** is 0 (update clock), mode should be 0 for all other devices except for MIO E Series devices. For these devices, **mode** is used to indicate the time of change of update rate, when a waveform is already in progress. If no waveform is in progress, **mode** is ignored. Set argument **mode** to 0 to indicate that you wish to change the update rate immediately. Set argument **mode** to 2 to indicate that you wish to change the update rate at the end of the current buffer. For MIO E Series devices you cannot change the update rate when using FIFO pulsed waveform generation and waveform is already in progress.

WFM_ClockRate

Continued

whichclock = 1:
When **whichclock** is 1 (delay clock), mode indicates whether delay clock should be enabled or disabled. When **mode** is 1, NI-DAQ will enable the delay clock. If you want to use FIFO pulse-waveform generation, you must set **mode** to 1. Notice that, if you enable delay clock, you must load finite iterations. If you load infinite iterations, NI-DAQ returns error code **fifoModeError**.

whichclock = 2:
mode is ignored in this case.

whichclock = 3:
Mode is ignored in this case.

If any of these conditions is not met, NI-DAQ will return **updateRateChangeError**.

Using This Function

You can calculate the actual update rate in seconds from the timebase resolution selected by **timebase** and **interval**, as shown by the following example.

Suppose that **timebase** equals 2. On an MIO device, this value selects the 100 kHz internal clock signal, which provides counter 2 with a rising edge to count every 10 µs, thus selecting the 10 µs resolution. On Lab and 1200 Series analog output devices, if the total update interval given by (timebase resolution) ∗ **interval** is greater than 65,535 µs, it programs counter B0 (if it is not busy in a data acquisition or a counting operation) to produce a clock of 100 kHz, which is used by the counter producing the update interval.

Also suppose that **interval** equals 25. This value indicates that counter 2 must count 25 rising edges of its input clock signal before issuing a request to produce a new voltage at the analog output channels.

The actual update rate in seconds is then 25 ∗ 10 µs = 250 µs. Thus, a new voltage is produced at the output channels every 250 µs.

The frequency of a waveform is related to the update rate and the number of points in the buffer (indicated in an earlier call to `WFM_Load`) as follows:

frequency = 1/(update rate ∗ points in the buffer)

WFM_ClockRate

Continued

You can make repeated calls to `WFM_ClockRate` to change the update rate of a waveform in progress. For MIO E Series devices, you can change the update interval immediately or at the end of current buffer. You cannot change the internal timebase already being used by the device, only the interval, and the following conditions must be met:

- **whichclock** is 0.
- You are not using FIFO pulsed waveform generation.
- The timebase has the value it had when you called this function before starting the waveform generation.
- At least one update was performed using the previously selected update interval if you want to change the interval immediately; i.e., when **mode** = 0.

If any of these conditions is not met, NI-DAQ will return **updateRateChangeError**.

To perform FIFO pulse waveform generation on an MIO E Series device, you must use the same timebase for update and delay clock. You must specify the delay time as the product of four number:

delay time = timebase period * delay interval * delay interval prescalar 1 * delay interval prescalar 2.

In this formula,

- Timebase period is a single period corresponding to the selected timebase (e.g., 50 ns when the 20 MHz clock is used)
- Delay interval corresponds to the interval argument in this function when **whichclock** = 1.
- Delay interval prescalar 1 corresponds to the interval argument you use in this function when **whichclock** = 2. If you do not call this function with **whichclock** = 2, this interval will be 1.
- Delay interval prescalar 2 corresponds to the interval argument you use in this function when **whichclock** = 3. If you do not call this function with **whichclock** = 3, this interval will be 2.

WFM_ClockRate

Continued

When **whichclock** = 2, NI-DAQ will ignore timebase and mode arguments. Legal range for delay interval prescalar 1 is 1 through 2^{24}.

When **whichclock** = 3, NI-DAQ will ignore timebase and mode arguments. Legal range for delay interval prescalar 2 is 2 through 2^{24}.

Example:

Let us compute the delay time after the following sequence of function calls:

```
WFM_ClockRate(deviceNumber, group, 0, -3, 1000, 0)
WFM_ClockRate(deviceNumber, group, 1, -3, 4000, 1)
WFM_ClockRate(deviceNumber, group, 2, -3, 7000, 1)
```

In this case, timebase period is 50 ns, delay interval is 4000, delay interval prescalar 1 is 7000, delay interval prescalar 2 is 2, so the delay time is
50 ns • 4000 • 7000 • 2 = 2,800,000,000 ns = 2.8 s.

Notice that the maximum possible delay time with the 20 MHz interval timebase is
50 ns • 2^{24} • 2^{24} • 2^{24} = 7.5 million years.

WFM_DB_Config

Format

status = WFM_DB_Config (deviceNumber, numChans, chanVect, dbMode, oldDataStop, partialTransferStop)

Purpose

Enables and disables the double-buffered mode of waveform generation.

Parameters

Input

Name	Type		Description
	Windows	Windows NT	
deviceNumber	I16	I32	assigned by configuration utility
numChans	I16	I32	number of analog output channels
chanVect	[I16]	[I32]	channel numbers
dbMode	I16	I32	enables or disables the double-buffered mode
oldDataStop	I16	I32	allow or disallow regeneration of data
partialTransferStop	I16	I32	whether to stop when a partial half buffer is transferred

Parameter Discussion

numchans indicates the number of analog output channels specified in the array **chanVect**.

Range:	1 or 2 for most devices.
	1 through 6 for AT-AO-6.
	1 through 10 for AT-AO-10.

WFM_DB_Config

Continued

chanVect is the user array of channel numbers indicating which analog output channels are to be checked to see if the next half buffer for that channel is available.
Channel number range:

0 or 1 for most devices.
0 through 5 for AT-AO-6.
0 through 9 for AT-AO-10.

dbMode is a flag whose value either enables or disables the double-buffered mode of waveform generation.

0:	Double-buffered mode disabled.
1:	Double-buffered mode enabled.

oldDataStop is a flag whose value enables or disables the mechanism whereby NI-DAQ stops the waveform generation when it is about to generate old data (data that has already been generated) a second time. Setting **oldDataStop** to 1 ensures seamless double-buffered waveform generation.

0:	Allow regeneration of data.
1:	Disallow regeneration of data.

partialTransferStop is a flag indicating whether to stop waveform generation when NI-DAQ transfers a partial half buffer to the analog output buffer using a `WFM_DB_Transfer` or `WFM_DB_StrTransfer` call. NI-DAQ will stop the waveform once NI-DAQ has output the partial half buffer.

0:	Allow partial half buffer transfers.
1:	Stop waveform generation after partial half buffer transfers.

Using This Function

Use `WFM_DB_Config` to turn double-buffered waveform generation on and off. With the double-buffered mode enabled, you can use `WFM_DB_Transfer` to transfer new data into the waveform buffer (selected by `WFM_Load`) as NI-DAQ generates the waveform. Because of the extra bookkeeping involved, unless you are going to use `WFM_DB_Transfer`, you should leave double buffering disabled. Refer to Chapter 5, *NI-DAQ Double Buffering*, of the *NI-DAQ User Manual for PC Compatibles* for a detailed discussion of double buffering.

If you are using an AT-AO-6/10, AT-MIO-16X, or AT-MIO-64F-5, enabling **partialTransfer** (or **oldDataStop**) will cause an artificial split in the waveform buffer, which requires DMA reprogramming at the end of each half buffer. Therefore, you should only enable these options if necessary.

WFM_DB_Config

Continued

(AT-AO-6/10 only) For double-buffered waveform generation with group 1 channels using DMA: If **oldDataStop** is enabled, partial half buffer transfers (using `WFM_DB_Transfer` or `WFM_DB_StrTransfer` calls) are only allowed if **partialTransferStop** is enabled.

For double-buffered waveform generation with group 1: The total number of points for all the group 1 channels (specified in `WFM_Load`) should be at least twice the size of the FIFO. Refer to the *AT-AO-6/10 User Manual* for information on the AT-AO-6/10 FIFO size.

(AT-MIO-16F-5 only) When using the double-buffered waveform generation and **oldDataStop** mode is enabled, the driver can alter bit 15 of the data points in the waveform buffer.

WFM_DB_HalfReady

Format

status = WFM_DB_HalfReady (deviceNumber, numChans, chanVect, halfReady)

Purpose

Checks if the next half buffer for one or more channels is available for new data during a double-buffered waveform generation operation. You can use `WFM_DB_HalfReady` to avoid the waiting period that can occur with the double-buffered transfer functions.

Parameters

Input

Name	Type		Description
	Windows	Windows NT	
deviceNumber	I16	I32	assigned by configuration utility
numChans	I16	I32	number of analog output channels
chanVect	[I16]	[I32]	channel numbers

Output

Name	Type		Description
	Windows	Windows NT	
halfReady	I16	I32	whether the next half buffer is available for new data

Parameter Discussion

numchans indicates the number of analog output channels specified in the array **chanVect**.

WFM_DB_HalfReady

Continued

chanVect is the user array of channel numbers indicating which analog output channels are to be checked to see if the next half buffer for that channel is available.
Channel number range:

0 or 1 for most devices.
0 through 5 for AT-AO-6.
0 through 9 for AT-AO-10.

halfReady indicates whether the next half buffer for all of the channels specified in **chanVect** is available for new data. When **halfReady** equals 1, you can use `WFM_DB_Transfer` or `WFM_DB_StrTransfer` to write new data to the next half buffer(s) immediately. When **halfReady** equals 0, the next half buffer for one or more channels is not ready for new data.

☞ **Note:** ***C Programmers*—halfReady *is a pass-by-reference parameter.***

Using This Function

Double-buffered waveform generation functions cyclically output data from the waveform buffer (specified in `WFM_Load`). The waveform buffer is divided into two halves so that NI-DAQ can write data from one half of the buffer to the output channels while filling the other half of the buffer with new data. This mechanism makes it necessary to alternately write to both halves of the waveform buffer so that NI-DAQ does not output the old data. Use `WFM_DB_Transfer` or `WFM_DB_StrTransfer` to transfer new data to a waveform buffer half. Both of these functions, when called, wait until NI-DAQ can complete the data transfer before returning. During slower paced waveform generation operations, this waiting period can be significant. You can use `WFM_DB_HalfReady` to call the transfer functions only when NI-DAQ can make the transfer immediately.

Refer to Chapter 5, *NI-DAQ Double Buffering*, in the *NI-DAQ User Manual for PC Compatibles* for an explanation of double buffering.

WFM_DB_StrTransfer

Format

status = WFM_DB_StrTransfer (deviceNumber, numChans, chanVect, strBuffer, count)

Purpose

Transfers new data from a character buffer into one or more waveform buffers (selected in `WFM_Load`) as waveform generation is in progress. `WFM_DB_StrTransfer` will wait until NI-DAQ can transfer data from the character buffer to the waveform buffer(s). `WFM_DB_StrTransfer` is intended for applications requiring a character or string buffer, such as reading from a file using the BASIC function, `Get`.

Parameters

Input

Name	Type		Description
	Windows	Windows NT	
deviceNumber	I16	I32	assigned by configuration utility
numChans	I16	I32	number of analog output channels
chanVect	[I16]	[I32]	channel numbers
count	U32	U32	number of new data points

WFM_DB_StrTransfer

Continued

Output

Name	Type		Description
	Windows	Windows NT	
strBuffer	[U8]	[U8]	new data that is to be transferred

Parameter Discussion

numchans indicates the number of analog output channels specified in the array **chanVect**.
Range: 1 or 2 for most devices.
1 through 6 for AT-AO-6.
1 through 10 for AT-AO-10.

chanVect is the user array of channel numbers indicating which analog output channels are to receive new data from the buffer.
Channel number range:
0 or 1 for most devices.
0 through 5 for AT-AO-6.
0 through 9 for AT-AO-10.

buffer is the array of new data that is to be transferred into the waveform buffer(s). `WFM_DB_StrTransfer` can transfer new data to more than one waveform buffer. For example, if two channels use separate waveform buffers (you called `WFM_Load` once for each channel), you can use a single call to `WFM_DB_StrTransfer` to transfer data to both waveform buffers. If **numChans** is greater than 1, NI-DAQ must interleave the data in **buffer** and data for each channel must follow the order given in **chanVect**.

count holds the number of new data points contained in **buffer**. When you make repeated calls to `WFM_DB_StrTransfer` during a waveform generation, it is most efficient if the amount of data transferred for each channel is equal to one-half the number of data points for the channel in the channel waveform buffer. For example, suppose channel 0 is using a waveform buffer of size 100 and channel 1 is using a waveform buffer of size 100. `WFM_DB_StrTransfer` should transfer 50 to channel 0 and 50 to channel 1, giving **count** a value of 100. If NI-DAQ makes transfers to more than one waveform buffer, it is most efficient if all the waveform buffers contain the same number of samples for each channel.

WFM_DB_StrTransfer

Continued

(AT-AO-6/10 only) For group 1 channels using DMA: If you enable **oldDataStop**, then transfers of less than half the number of samples in the circular waveform buffer are only allowed if **partialTransferStop** is enabled.

Using This Function

Use `WFM_DB_StrTransfer` to transfer new data into one or more waveform buffers as waveform generation is in progress. The double-buffered mode, with **oldDataStop** set to 1, ensures that each data point for a specified output channel is generated exactly once. If you enable **partialTransferStop**, a transfer of less than half of the waveform buffer size of a channel will stop the waveform generation once NI-DAQ has output the partial half buffer.

(AT-MIO-16F-5 only) If the waveform buffer that you used in calling `WFM_Load` was aligned by calling `Align_DMA_Buffer`, `WFM_DB_StrTransfer` automatically indexes to the correct starting index in the waveform buffer, if necessary. You do not need to align the buffer used in the `WFM_DB_StrTransfer` call.

WFM_DB_Transfer

Format

status = WFM_DB_Transfer (deviceNumber, numChans, chanVect, buffer, count)

Purpose

Transfers new data into one or more waveform buffers (selected in `WFM_Load`) as waveform generation is in progress. `WFM_DB_Transfer` will wait until NI-DAQ can transfer data from the buffer to the waveform buffer(s).

Parameters

Input

Name	Type		Description
	Windows	Windows NT	
deviceNumber	I16	I32	assigned by configuration utility
numChans	I16	I32	number of analog output channels
chanVect	[I16]	[I32]	channel numbers
count	U32	U32	number of new data points

Output

Name	Type		Description
	Windows	Windows NT	
buffer	[I16], [F32], HDL	[I16], [F32], HDL	new data that is to be transferred

WFM_DB_Transfer

Continued

Parameter Discussion

numchans indicates the number of analog output channels specified in the array **chanVect**.

Range:	1 or 2 for most devices.
	1 through 6 for AT-AO-6.
	1 through 10 for AT-AO-10.

chanVect is the user array of channel numbers indicating which analog output channels are to receive new data from the buffer.
Channel number range:

0 or 1 for most devices.
0 through 5 for AT-AO-6.
0 through 9 for AT-AO-10.

buffer is the array of new data that is to be transferred into the waveform buffer(s). `WFM_DB_Transfer` can transfer new data to more than one waveform buffer. For example, if two channels use separate waveform buffers (you called `WFM_Load` once for each channel), you can use a single call to `WFM_DB_Transfer` to transfer data to both waveform buffers. If **numChans** is greater than 1, NI-DAQ must interleave the data in **buffer** and data for each channel must follow the order given in **chanVect**. For any device, **buffer** can be a PC memory address or an `NI_DAQ_Mem` array. See the `NI_DAQ_Mem_Alloc` function in this manual or the *Allocating DSP Board Memory* section in the *NI-DSP User Manual for PC Compatibles* for more information.

count holds the number of new data points contained in **buffer**. When you make repeated calls to `WFM_DB_Transfer` during a waveform generation, it is most efficient if the amount of data transferred for each channel is equal to one-half the number of data points for the channel in the channel's waveform buffer. For example, suppose channel 0 is using a waveform buffer of size 100 and channel 1 is using a waveform buffer of size 100. `WFM_DB_Transfer` should transfer 50 to channel 0 and 50 to channel 1, giving **count** a value of 100. If NI-DAQ makes transfers to more than one waveform buffer, it is most efficient if all the waveform buffers contain the same number of samples for each channel.

(AT-AO-6/10 only) For group 1 channels using DMA, if you enable **oldDataStop**, transfers of less than half the number of samples in the circular waveform buffer are only allowed if you enable **partialTransferStop**.

WFM_DB_Transfer

Continued

Using This Function

Use `WFM_DB_Transfer` to transfer new data into one or more waveform buffers as waveform generation is in progress. The double-buffered mode, with **oldDataStop** set to 1, ensures that NI-DAQ generates each data point for a specified output channel exactly once. If you enabled **partialTransferStop**, a transfer of less than half of the waveform buffer size of a channel will stop the waveform generation once NI-DAQ has output the partial half buffer.

(AT-MIO-16F-5 only) If the waveform buffer that you used in calling `WFM_Load` was aligned by calling `Align_DMA_Buffer`, `WFM_DB_Transfer` automatically indexes to the correct starting index in the waveform buffer, if necessary. You need not align the buffer used in the `WFM_DB_Transfer` call.

WFM_from_Disk

Format

status = WFM_from_Disk (deviceNumber, numChans, chanVect, fileName, startPt, endPt, iterations, rate)

Purpose

Assigns a disk file to one or more analog output channels, selects the rate and the number of times the data in the file is to be generated, and starts the generation. `WFM_from_Disk` always waits for completion before returning, unless you call `Timeout_Config`.

Parameters

Input

Name	Type		Description
	Windows	Windows NT	
deviceNumber	I16	I32	assigned by configuration utility
numChans	I16	I32	number of analog output channels
chanVect	[I16]	[I32]	channel numbers
fileName	STR	STR	name of the data file containing the waveform data
startPt	U32	U32	place in a file where waveform generation is to begin
endPt	U32	U32	place in a file where waveform generation is to end
iterations	U32	U32	number of times generated
rate	F64	F64	desired rate in points per second

WFM_from_Disk

Continued

Parameter Discussion

numchans indicates the number of analog output channels specified in the array **chanVect**.

chanVect is the user array of channel numbers indicating which analog output channels are to receive output data from the file.
Channel number range:
0 or 1 for most devices.
0 through 5 for AT-AO-6.
0 through 9 for AT-AO-10.

fileName is the name of the data file containing the waveform data. For MIO devices (except AT-MIO-16X), AT-AO-6/10, and Lab and 1200 Series analog output devices, the file must contain integer data ranging from 0 to 4,095 for unipolar mode and from -2,048 to 2,047 for bipolar mode. For an AT-MIO-16X, the file must contain integer data ranging from 0 to 65,535 for unipolar mode, and from -32,768 to +32,767 for bipolar mode. For an AT-DSP2200, the file can contain integer or single-precision floating-point data (see `DSP2200_Config`). If **numChans** is greater than one, you must interleave the data values in the file in ascending channel order.

startPt is the place in a file where waveform generation is to begin.
Range: 1 through the number of samples in the file.

endPt is the place in a file where waveform generation is to end. A value of 0 for **endPt** has a special meaning. When **endPt** equals 0, waveform generation will proceed to the end of the file and wrap around to **startPt** if **iterations** is greater than 1.
Range: 1 through the number of samples in the file.

iterations is the number of times the data in the file is generated.
Range: 1 through 2^{32} - 1.

WFM_from_Disk

Continued

rate is the rate of waveform generation you want in points per second (pts/s). A value of 0.0 for **rate** means that external update pulses (applied to OUT2 for the MIO-16 and AT-MIO-16D, to EXTDACUPDATE for the AT-MIO-16F-5, AT-MIO-64F-5, and AT-MIO-16X, and to EXTUPDATE for the AT-AO-6/10 and Lab and 1200 Series analog output devices) will determine the waveform generation rate. The AT-DSP2200 does not support external update pulses. If you are using an MIO E Series device, see the `Select_Signal` function for information about the external timing signals.

Range: 0.0 for external update or approximately 0.0015 to 500,000 pts/s. Your maximum rate depends on your device type and your computer system. If you have an AT-DSP2200, there are exactly 16 different rates are available. They are (in points per second): 51,200, 48,000, 44,100, 32,000, 25,600, 24,000, 22,050, 16,000, 12,800, 12,000, 11,025, 8,000, 6,400, 6,000, 5,513, and 4,000.

If the number of points that represent one cycle of the waveform equals **count**, the frequency of the generated waveform is related to the **rate** by the following formula:

frequency = (**rate**/**count**) cycles/s

Using This Function

`WFM_from_Disk` initiates a waveform generation operation. NI-DAQ writes the portion of data in the file determined by **startPt** and **endPt** to the specified analog output channels at a rate as close to the rate you want as the hardware permits (see `WFM_Rate` for further explanation). If **numChans** is greater than one, NI-DAQ writes the data values from file to the DAC in ascending order. `WFM_from_Disk` always waits until the requested number of file iterations is complete before returning.

If you have changed the analog output configuration from the defaults by changing the jumpers on the device, you must call `AO_Configure` to set the software copies of the settings prior to calling `WFM_from_Disk`.

NI-DAQ ignores the group settings made by calling `WFM_Group_Setup` when you call `WFM_from_Disk` and are not changed after you execute `WFM_from_Disk`.

☞ **Note:** ***For the MIO-16 and AT-MIO-16D, counter 2 must be available in order to use waveform generation. If an interval scan is in progress (see*** `SCAN_Start`***) or a CTR function is using counter 2, waveform generation cannot proceed.***

WFM_from_Disk

Continued

For the AT-MIO-16F-5, AT-MIO-64F-5, and AT-MIO-16X, you can use counters 1, 2, and 5, as well as a dedicated external update signal, to generate either interrupt of DMA requests. If you use counter 1 or 2, a RTSI line must also be available. NI-DAQ uses the first available counter among counters 5, 2, and 1, in that order.

For Lab and 1200 Series analog output devices, if the rate is smaller than 15.26 pts/s and counter B0 is busy in a data acquisition or counting operation, waveform generation cannot proceed.

On Am9513-based devices, if you want to externally trigger a waveform generation operation, you can do so by first changing the gating mode of the counter NI-DAQ will use.

`WFM_from_Disk` will use either the default gating mode (none) or the gating mode you specify through the `CTR_Config` function. You will need to connect your trigger signal to the gate pin on the I/O connector. Refer to the `CTR_Config` function description for details.

On a variety of MIO E Series devices, you can externally trigger a waveform generation in a variety of ways. Refer to the `Select_Signal` function for more details.

WFM_Group_Control

Format

status = WFM_Group_Control (deviceNumber, group, operation)

Purpose

Controls waveform generation for a group of analog output channels.

Parameters

Input

Name	Type		Description
	Windows	Windows NT	
deviceNumber	I16	I32	assigned by configuration utility
group	I16	I32	group of analog output channels
operation	I16	I32	operation to be performed

Parameter Discussion

group is the group of analog output channels (see `WFM_Group_Setup`).

Range:	1 for most devices.
	1 or 2 for the AT-AO-6/10.

operation selects the operation to be performed for the group of output channels.

0 (CLEAR):	Terminates a waveform operation for the group of analog output channels. The last voltage produced at the DAC is maintained indefinitely. After you execute CLEAR for an analog output group, you must call `WFM_Load` before NI-DAQ can restart waveform generation using **operation** = START. (AT-MIO-16F-5 only) If you aligned the data buffer used in the waveform generation by calling `Align_DMA_Buffer`, CLEAR unaligns the buffer. That is, the data samples start at index 0 of the buffer. If

WFM_Group_Control

Continued

you want to use the same buffer again for waveform generation, you must call `Align_DMA_Buffer` again before calling `WFM_Load`.

1 (START): Initiates waveform generation at the analog output channels in group. Select the waveform generation update rate for the group in `WFM_ClockRate`. Your application must call **operation** = CLEAR before terminating, if you execute START. If you do not execute CLEAR, unpredictable behavior may result.

☞ **Notes:** ***For the MIO-16 and AT-MIO-16D, counter 2 must be available in order to use waveform generation. If an interval scan is in progress (see*** `SCAN_Start`***) or a*** `CTR` ***function is using counter 2, waveform generation cannot proceed.***

For Lab and 1200 Series analog output devices, if the rate is smaller than 15.26 pts/s and counter B0 is busy in a data acquisition operation, waveform generation cannot proceed.

For the AT-MIO-16F-5, AT-MIO-64F-5, and AT-MIO-16X, one of the counters 1, 2, or 5 must be available. NI-DAQ uses the first available counter among counters 5, 2, and 1, in that order. If counter 5 is in use, and NI-DAQ if forced to use counters 2 or 1, a RTSI line must also be available.

On MIO devices, if you wish to externally trigger a waveform generation operation, you can do so by first changing the gating mode of the counter NI-DAQ will use.

`WFM_OP` ***will use either the default gating mode (none) or the gating mode you specify through the*** `CTR_Config` ***function. You will need to connect your trigger signal to the gate pin on the I/O connector.***

2 (PAUSE): Temporarily halts waveform generation for the group of channels. NI-DAQ maintains the last voltage written to the DAC indefinitely.

☞ **Note:** ***This value of*** **operation=2** ***is not supported for the MIO E Series devices.***

WFM_Group_Control

Continued

4 (RESUME):	Restarts waveform generation for the group of channels previously halted by **operation** = PAUSE.

☞ **Note:** ***This value of* operation=4 *is not supported for the MIO E Series devices.***

When you have halted a waveform generation by executing PAUSE, RESUME restarts the waveform exactly at the point in your buffer where it left off. If *n* iterations of the buffer remained to be done when you executed **operation** = PAUSE, NI-DAQ generates those *n* iterations after you execute RESUME. RESUME will restart waveform generation if the number of iterations specified in `WFM_Load` has completed.

WFM_Group_Setup

Format

status = WFM_Group_Setup (deviceNumber, numChans, chanVect, group)

Purpose

Assigns one or more analog output channels to a waveform generation group. A call to `WFM_Group_Setup` is only required for the AT-AO-6/10. By default, both analog output channels for the MIO devices, Lab-PC+, and AT-DSP2200 are in group 1.

Parameters

Input

Name	Type		Description
	Windows	Windows NT	
deviceNumber	I16	I32	assigned by configuration utility
numChans	I16	I32	number of analog output channels
chanVect	[I16]	[I32]	channel numbers
group	I16	I32	group number

Parameter Discussion

numchans indicates the number of analog output channels specified in the array **chanVect**. Zero clears the channel assignments for group.

Range: 0 through 2 for most devices.
0 through 6 for AT-AO-6.
0 through 10 for AT-AO-10.

chanVect is your array of channel numbers indicating which analog output channels are in a group.

Channel number range:
0 or 1 for most devices.
0 through 5 for AT-AO-6.

WFM_Group_Setup

Continued

	0 through 9 for AT-AO-10.

group is the group number.

Range:	1 for most devices.
	1 or 2 for AT-AO-6/10.

Using This Function

For the AT-AO-6/10, you can assign analog output channels to one of two waveform generation groups. Each group has a separate update clock source. You can assign different update rates to each group by calling `WFM_ClockRate`.

There are two restrictions placed on group 1 channel assignments for the AT-AO-6/10. If you assign more than one output channel to group 1, you must number the channels consecutively 0 through *N* (where *N* is the highest channel number in group 1).

On the AT-AO-10, when you assign a single channel to group 1, only channels 0 through 7 are eligible. You can only assign channels 8 and 9 to group 2.

WFM_Load

Format

status = WFM_Load (deviceNumber, numChans, chanVect, buffer, count, iterations, mode)

Purpose

Assigns a waveform buffer to one or more analog output channels and indicates the number of waveform cycles to generate.

Parameters

Input

Name	Type		Description
	Windows	Windows NT	
deviceNumber	I16	I32	assigned by configuration utility
numChans	I16	I32	number of analog output channels
chanVect	[I16]	[I32]	channel numbers
buffer	[I16], [F32], HDL	[I16], [F32], HDL	values that are converted to voltages by DACs
count	U32	U32	number of points in the buffer
iterations	U32	U32	number of times the waveform generation steps through **buffer**
mode	I16	I32	enables or disables FIFO mode

WFM_Load

Continued

Parameter Discussion

numchans indicates the number of analog output channels specified in the array **chanVect**.

Range:	1 or 2 for most devices.
	1 through 6 for AT-AO-6.
	1 through 10 for AT-AO-10.

chanVect is the user array of channel numbers indicating which analog output channels the buffer is to be assigned.

Channel number range:

0 or 1 for most devices.
0 through 5 for AT-AO-6.
0 through 9 for AT-AO-10.

buffer is an array of values that are converted to voltages by DACs. For MIO devices (except AT-MIO-16X), AT-AO-6/10, and Lab and 1200 Series analog output devices, **buffer** must contain integer data. These data will range from 0 to 4,095 in unipolar mode and from -2,048 to 2,047 in bipolar mode. For an AT-MIO-16X, data will range from 0 to 65,535 in unipolar mode and from -32,768 to +32,767 in bipolar mode. For an AT-DSP2200, **buffer** can contain integer or single-precision floating-point data (see `DSP2200_Config`). For any device, **buffer** can be a PC memory address or an `NI_DAQ_Mem` array. See the `NI_DAQ_Mem_Alloc` function in this manual or the *Allocating DSP Board Memory* section in the *NI-DSP User Manual for PC Compatibles* for more information.

count is the number of points in your buffer. When you use interleaved waveform generation, **count** should be a multiple of **numChans** and not less than 2 * **numChans**. When you use double-buffered interleaved waveform generation, **count** should not be less than 4 * **numChans**.

Range:	1 through 2^{32} - 1 (except MIO E Series devices).
	2 through 2^{24} (MIO E Series devices).

iterations is the number of times the waveform generation steps through **buffer**. A value of 0 means that waveform generation proceeds indefinitely.

Range:	0 through 2^{32} - 1.

Enabling FIFO mode waveform generation places some restrictions on the allowable values for the **iterations** parameter. Please refer to the **mode** parameter description below.

WFM_Load

Continued

Enabling pulsed FIFO mode waveform generation by turning on the delay clock via `WFM_ClockRate` places two additional restrictions on the allowed values of **iterations** and also changes its meaning. Setting **iterations** to 0 is not allowed. Setting **iterations** to 1 is not allowed if you are using an AT-MIO-16X or AT-MIO-64F-5. Also, instead of determining the number of times the waveform generation steps through **buffer** before stopping, pulsed FIFO mode causes the **iterations** setting to determine the number of times the data in the FIFO is generated before pausing for the specified delay. Once the delay has elapsed, the data in the FIFO is generated again. In other words, when you use pulsed FIFO mode, the value of **iterations** determines the number of cycles through the FIFO that will occur between delays, and the pattern of waveform followed by delay followed by waveform etc. goes on indefinitely.

mode allows you to indicate whether to use FIFO mode waveform generation.

Range:	0 or 1 for the AT-MIO-16E-2, AT-MIO-64E-3, NEC-MIO-16E-4, AT-MIO-16X, AT-MIO-64F-5, and AT-AO-6/10.
	0 for all other devices.

When **mode** is 0, NI-DAQ does not use FIFO mode waveform generation. When **mode** is 1 and all of the following conditions are satisfied, NI-DAQ will use FIFO mode waveform generation:

- The waveform buffer is small enough to reside in the DAC FIFO. The size of the DAC FIFO is 2,048 points on the AT-MIO-16E-2, AT-MIO-64E-3, NEC-MIO-16E-4, AT-MIO-16X, and AT-MIO-64F-5, and 1,024 points on the AT-AO-6/10. If you load more than one channel, the total number of points must be less than or equal to the FIFO size.
- You have not enabled double-buffered waveform generation mode.
- For the AT-AO-6/10, iterations must be 0. For the AT-MIO-16X and AT-MIO-64F-5, iterations can be:
  - 0 for continuous cyclic waveform generation.
  - 1 through 65,535 inclusive for programmed cyclic waveform generation.
  - 2 through 65,535 inclusive for pulsed waveform generation.
- All the channels listed in **chanVect** must belong to group 1.
- If more than one channel of group 1 is loaded, the number of points per channel and iterations are the same for each channel. Also, all the channels of group 1 must have the same **mode**.

WFM_Load

Continued

NI-DAQ returns error **fifoModeError** if any of the previously described conditions is not satisfied and **mode** is 1. If you call the `WFM_Load` function several times to load different channels, the `WFM_Group_Control` function will check for conditions 1 and 5.

When **mode** is 1 and you have enabled the delay clock (see the `WFM_Clock_Config` function), the waveform generation proceeds until it is stopped by software. In this case, **iterations** indicates how many times the waveform will be generated between delays.

Using This Function

`WFM_Load` assigns your buffer to a selected analog output channel or channels. The values in this buffer are translated to voltages by the digital-to-analog (D/A) circuitry and produced at the output channel when you have called `WFM_Group_Control` (**operation** = START) for a channel group. If you have changed the analog output configuration from the defaults by changing the jumpers on the device, you must call `AO_Configure` to set the software copies of the settings prior to calling `WFM_Group_Control` (**operation** = START). You can make repeated calls to `WFM_Load` to change the shape of a waveform in progress, except on MIO E Series devices and SCXI DAQ modules used with Remote SCXI; if you make repeated calls using these devices, this function will return a **transferInProgError**. You must also use the parameter values for **numChans** and **chanVect** used in the call to `WFM_Load` prior to starting the waveform when making calls to `WFM_Load` while a waveform is in progress.

(AT-MIO-16F-5 only) If buffer has been aligned by a previous call to `Align_DMA_Buffer`, `WFM_Load` automatically indexes into the buffer to the new starting point of the data. If you call `WFM_Load` with a new buffer while a waveform generation is in progress, NI-DAQ unaligns the previous buffer when the function returns.

WFM_Op

Format

status = WFM_Op (deviceNumber, numChans, chanVect, buffer, count, iterations, rate)

Purpose

Assigns a waveform buffer to one or more analog output channels, selects the rate and the number of times the data in the buffer is to be generated, and starts the generation. If the number of buffer generations is finite, `WFM_Op` waits for completion before returning, unless you call `Timeout_Config`.

Parameters

Input

Name	Type		Description
	Windows	Windows NT	
deviceNumber	I16	I32	assigned by configuration utility
numChans	I16	I32	number of analog output channels
chanVect	[I16]	[I32]	channel numbers
buffer	[I16], [F32], HDL	[I16], [F32], HDL	values that are converted to voltages by DACs
count	U32	U32	number of points in the buffer
iterations	U32	U32	number of times the waveform generation steps through **buffer**
rate	F64	F64	desired rate in pts/s

WFM_Op

Continued

Parameter Discussion

numchans indicates the number of analog output channels specified in the array **chanVect**.

chanVect is the user array of channel numbers indicating which analog output channels are to receive output data from the buffer.
Channel number range:
0 or 1 for most devices.
0 through 5 for AT-AO-6.
0 through 9 for AT-AO-10.

buffer is an array of values that DACs convert to voltages. For MIO devices (except AT-MIO-16X), AT-AO-6/10, and Lab and 1200 Series analog output devices, **buffer** must contain integer data. These data will range from 0 to 4,095 in unipolar mode and from -2,048 to 2,047 in bipolar mode. For an AT-MIO-16X, data will range from 0 to 65,535 in unipolar mode and from -32,768 to +32,767 in bipolar mode. For an AT-DSP2200, **buffer** can contain integer or single-precision floating-point data (see `DSP2200_Config`). For any device, **buffer** can be a PC memory address or an `NI_DAQ_Mem` array. See the `NI_DAQ_Mem_Alloc` function in this manual or the *Allocating DSP Board Memory* section in the *NI-DSP User Manual for PC Compatibles* for more information.

count is the number of points in your buffer. When using interleaved waveform generation, **count** should be a multiple of **numChans** and not less than 2 * **numChans**.
Range: 1 through 2^{32} - 1 (except MIO E Series devices).
2 through 2^{24} (MIO E Series devices).

iterations is the number of times the waveform generation steps through **buffer**. A value of 0 means that waveform generation proceeds indefinitely.
Range: 0 through 2^{32} - 1.

WFM_Op

Continued

rate is the rate of waveform generation you want in points per second (pts/s). A value of 0.0 for **rate** means that external update pulses (applied to OUT2 for the MIO-16 and AT-MIO-16D, to EXTDACUPDATE for the AT-MIO-16F-5, AT-MIO-64F-5, and AT-MIO-16X, and to EXTUPDATE for the AT-AO-6/10 and Lab and 1200 Series analog output devices) will determine the waveform generation rate. The AT-DSP2200 does not support external update pulses.

Range: 0.0 for external update or approximately 0.0015 to 500,000 pts/s. Your maximum **rate** depends on your device type and your computer system. If you have an AT-DSP2200 there are exactly 16 different rates are available. They are (in points per second): 51,200, 48,000, 44,100, 32,000, 25,600, 24,000, 22,050, 16,000, 12,800, 12,000, 11,025, 8,000, 6,400, 6,000, 5,513, and 4,000.

If the number of points that represent one cycle of the waveform equals **count**, the frequency of the generated waveform is related to the **rate** by the following formula:

frequency = (**rate**/**count**) cycles/s

Using This Function

`WFM_Op` initiates a waveform generation operation. NI-DAQ writes the data in the buffer to the specified analog output channels at a rate as close to the rate you want as the hardware permits (see `WFM_Rate` for a further explanation). With the exception of indefinite waveform generation, `WFM_Op` waits until the waveform generation is complete before returning (that is, it is synchronous).

(AT-MIO-16F-5 only) If you have aligned the buffer with a previous call to `Align_DMA_Buffer`, `WFM_Op` automatically indexes into the buffer at the new starting point if necessary. If the call to `WFM_Op` is synchronous, when the function returns, the buffer is unaligned. That is, the data samples will start at index 0 of the buffer. If the waveform generation is indefinite, the buffer remains aligned until you call `WFM_Group_Control` (**operation** = CLEAR).

If you have changed the analog output configuration from the defaults by changing the jumpers on the device, you must call `AO_Configure` to set the software copies of the settings prior to calling `WFM_Op`.

NI-DAQ ignores the group settings made by calling `WFM_Group_Setup` when you call `WFM_Op` and does not change the settings after you execute `WFM_Op`.

WFM_Op

Continued

☞ **Note:** ***For the MIO-16 and AT-MIO-16D, counter 2 must be available in order to use waveform generation. If an interval scan is in progress (see*** `SCAN_Start`***) or a CTR function is using counter 2, waveform generation cannot proceed.***

For the AT-MIO-16F-5, AT-MIO-64F-5, and AT-MIO-16X, you can use counter 1, 2, and 5, as well as a dedicated external update signal, to generate either interrupt or DMA requests. If you use counter 1 or 2, a RTSI line must also be available. NI-DAQ uses the first available counter among counters 5, 2, and 1, in that order.

For Lab and 1200 Series analog output devices, if the rate is smaller than 15.26 and counter B0 is busy in a data acquisition or counting operation, waveform generation cannot proceed.

On Am9513-based devices, if you want to externally trigger a waveform generation operation, you can do so by first changing the gating mode of the counter NI-DAQ will use.

`WFM_OP` will use either the default gating mode (none) or the gating mode you specify through the `CTR_Config` function. You will need to connect your trigger signal to the gate pin on the I/O connector. Refer to the `CTR_Config` function description for details.

On a variety of MIO E Series devices, you can externally trigger a waveform generation in a variety of ways. Refer to the `Select_Signal` function for more details.

WFM_Rate

Format

status = WFM_Rate (rate, units, timebase, updateInterval)

Purpose

Converts a waveform generation update rate into the timebase and update-interval values needed to produce the rate you want.

Parameters

Input

Name	**Type**		**Description**
	Windows	**Windows NT**	
rate	F64	F64	update rate you want
units	I16	I32	units used

Output

Name	**Type**		**Description**
	Windows	**Windows NT**	
timebase	I16	I32	resolution of clock signal
updateInterval	U32	U32	number of timebase units

Parameter Discussion

rate is the waveform generation update rate you want. **rate** is expressed in either pts/s or seconds per point (s/pt), depending on the value of the **units** parameter.

Range: Roughly 0.00153 pts/s through 500,000 pts/s or 655 s/pt through 0.000002 s/pt.

units indicates the units used to express **rate**.

0: pts/s.

WFM_Rate

Continued

1: s/pt.

timebase is a code representing the resolution of the onboard clock signal that the device uses to produce the update rate you want. You can input the value returned in **timebase** directly to `WFM_ClockRate`. **timebase** has the following possible values:

-3:	20 MHz clock used as a timebase (50 ms) (MIO E Series only).
-1:	5 MHz clock used as timebase (200 ns resolution) (AT-MIO-16F-5, AT-MIO-64F-5, and AT-MIO-16X only).
1:	1 MHz clock used as timebase (1 µs resolution).
2:	100 kHz clock used as timebase (10 µs resolution).
3:	10 kHz clock used as timebase (100 µs resolution).
4:	1 kHz clock used as timebase (1 ms resolution).
5:	100 Hz clock used as timebase (10 ms resolution).

updateInterval is the number of timebase units that elapse between consecutive writes (updates) to the D/A converters. The combination of the timebase resolution value and the **updateInterval** produces the waveform generation rate you want. You can input the value returned in **updateInterval** directly to `WFM_ClockRate`.
Range: 2 through 65,535.

☞ **Note:** ***If you are using an SCXI-1200 with remote SCXI, the maximum rate will depend on the baud rate setting and*** **updateRate*****. Refer to the*** *SCXI-1200 User Manual* ***for more details.***

☞ **Note:** ***C Programmers—*****timebase** ***and*** **updateInterval** ***are pass-by-reference parameters.***

Using This Function

`WFM_Rate` produces timebase and update-interval values to closely match the update rate you want. To calculate the actual rate produced by these values, first determine the clock resolution that corresponds to the value **timebase** returns. Then use the appropriate formula below, depending on the value specified for **units**:

units = 0 (pts/s).

actual rate = 1/(clock resolution ∗ **updateInterval**).

units = 1 (s/pt).

actual rate = clock resolution ∗ **updateInterval.**

This function does not support the AT-DSP2200.

WFM_Scale

Format

status = WFM_Scale (deviceNumber, chan, count, gain, voltArray, binArray)

Purpose

Translates an array of floating-point values that represent voltages into an array of binary values that produce those voltages when NI-DAQ writes the binary array to one of the DACs. This function uses the current analog output configuration settings to perform the conversions.

Parameters

Input

Name	Type		Description
	Windows	Windows NT	
deviceNumber	I16	I32	assigned by configuration utility
chan	I16	I32	analog output channel
count	U32	U32	number of points in **buffer**
gain	F64	F64	multiplier applied as the translation is performed
voltArray	[F64]	[F64]	input double-precision values

Output

Name	Type		Description
	Windows	Windows NT	
binArray	[I16]	[I16]	binary values converted from the voltages

WFM_Scale

Continued

Parameter Discussion

chan indicates to which analog output channel the binary array is to be assigned.

Range:	0 or 1 for most devices.
	0 through 5 for AT-AO-6.
	0 through 9 for AT-AO-10.

count is the number of points in your buffer.
Range: 1 through 2^{32} - 1.

gain is a multiplier applied to the array as NI-DAQ performs the translation. If the result of multiplying each element in the array by the value of **gain** produces a voltage that is out of range, NI-DAQ sets the voltage to the maximum or minimum value and returns an error. NI-DAQ still completes the translation, however.
Range: Any real number that produces a voltage within the analog output range.

voltArray is the input array of double-precision values that represents the voltages to be produced at one of the outputs.
Range: Any real number that produces a voltage within the analog output range.

binArray is the array of binary values converted from the voltages contained in **voltArray**. The values in **binArray** produce the original voltages when NI-DAQ writes them to a DAC on your device. Refer to Appendix B, *Voltage Calculation and Gain Settings*, for the calculation of binary value.

Using This Function

`WFM_Scale` calculates each binary value using the following formulas:

- Unipolar configuration:

  MIO-16, AT-MIO-16D, AT-MIO-16F-5, AT-MIO-64F-5, and AT-AO-6/10: **binVal** = **voltage** ∗ (gain ∗ (4,096/**outputRange**)).

  AT-MIO-16X: **binVal** = **voltage** ∗ (gain ∗ (65,536/**outputRange**)).

  Lab and 1200 Series analog output devices: **binVal** = **voltage** ∗ (gain ∗ (4,096/10.0)).

- Bipolar configuration:

  MIO-16, AT-MIO-16D, AT-MIO-16F-5, AT-MIO-64F-5, and AT-AO-6/10: **binVal** = **voltage** ∗ (gain ∗ (2,048/**outputRange**)).

  AT-MIO-16X: **binVal** = **voltage** ∗ (gain ∗ (32,768/**outputRange**)).

WFM_Scale

Continued

Lab and 1200 Series analog output devices: **binVal** = **voltage** ∗ (gain ∗ (2,048/5.0)).

AT-DSP2200: **binVal** = **voltage** ∗ (gain ∗ (32,768/2.828)).

Notice that **outputRange** is the value indicated in `AO_Configure`. Since you can independently configure the analog output channels for range and polarity (configured for channel pairs for the AT-AO-6/10), you can translate the same voltage to different 12-bit values for each channel, except for the AT-MIO-16X in which voltage is translated to 16-bit values. If, for some reason, you call `AO_Configure` and the function assigns a value of 0.0 V to **outputRange**, `WFM_Scale` returns a **invalidValueError**.

Appendix

A

Status Codes

This appendix lists the status codes returned by NI-DAQ, including the name and description.

Each NI-DAQ function returns a status code that indicates whether the function was performed successfully. When an NI-DAQ function returns a code that is a negative number, it means that the function did not execute. When a positive status code is returned, it means that the function did execute, but with a potentially serious side effect. A summary of the status codes is listed in Table A-1.

☞ **Note:** ***All status codes and descriptions are also listed in NI-DAQ online help.***

Table A-1. Status Code Summary

Status Code	Status Name	Description
-10001	**syntaxError**	An error was detected in the input string; the arrangement or ordering of the characters in the string is not consistent with the expected ordering.
-10002	**semanticsError**	An error was detected in the input string; the syntax of the string is correct, but certain values specified in the string are inconsistent with other values specified in the string.
-10003	**invalidValueError**	The value of a numeric parameter is invalid.
-10004	**valueConflictError**	The value of a numeric parameter is inconsistent with another one, and therefore the combination is invalid.
-10005	**DSPbadDeviceError**	The device is invalid.
-10006	**badLineError**	The line is invalid.

Table A-1. Status Code Summary (Continued)

Status Code	Status Name	Description
-10007	**badChanError**	A channel is out of range for the board type or input configuration; or the combination of channels is not allowed; or the scan order must be reversed (0 last).
-10008	**badGroupError**	The group is invalid.
-10009	**badCounterError**	The counter is invalid.
-10010	**badCountError**	The count is too small or too large for the specified counter; or the given I/O transfer count is not appropriate for the current buffer or channel configuration.
-10011	**badIntervalError**	The analog input scan rate is too fast for the number of channels and the channel clock rate; or the given clock rate is not supported by the associated counter channel or I/O channel.
-10012	**badRangeError**	The analog input or analog output voltage range is invalid for the specified channel.
-10013	**badErrorCodeError**	The driver returned an unrecognized or unlisted error code.
-10014	**groupTooLargeError**	The group size is too large for the board.
-10015	**badTimeLimitError**	The time limit is invalid.
-10016	**badReadCountError**	The read count is invalid.
-10017	**badReadModeError**	The read mode is invalid.
-10018	**badReadOffsetError**	The offset is unreachable.
-10019	**badClkFrequencyError**	The frequency is invalid.
-10020	**badTimebaseError**	The timebase is invalid.

Table A-1. Status Code Summary (Continued)

Status Code	Status Name	Description
-10021	**badLimitsError**	The limits are beyond the range of the board.
-10022	**badWriteCountError**	Your data array contains an incomplete update, or you are trying to write past the end of the internal buffer, or your output operation is continuous and the length of your array is not a multiple of one half of the internal buffer size.
-10023	**badWriteModeError**	The write mode is out of range or is disallowed.
-10024	**badWriteOffsetError**	Adding the write offset to the write mark places the write mark outside the internal buffer.
-10025	**limitsOutOfRangeError**	The requested input limits exceed the board's capability or configuration. Alternative limits were selected.
-10026	**badBufferSpecificationError**	The requested number of buffers or the buffer size is not allowed; e.g., Lab-PC buffer limit is 64K samples, or the board does not support multiple buffers.
-10027	**badDAQEventError**	For DAQEvents 0 and 1 general value A must be greater than 0 and less than the internal buffer size. If DMA is used for DAQEvent 1 general value A must divide the internal buffer size evenly, with no remainder. If the TIO-10 is used for DAQEvent 4 general value A must be 1 or 2.
-10028	**badFilterCutoffError**	The cutoff frequency specified is not valid for this device.
-10029	**obsoleteFunctionError**	The function you are calling is no longer supported in this version of the driver.

Table A-1. Status Code Summary (Continued)

Status Code	Status Name	Description
-10030	**badBaudRateError**	The specified baud rate for communicating with the serial port is not valid on this platform.
-10031	**badChassisIDError**	The specified SCXI chassis does not correspond to a configured SCXI chassis.
-10032	**badModuleSlotError**	The SCXI module slot that was specified is invalid or corresponds to an empty slot.
-10033	**invalidWinHandleError**	The window handle passed to the function is invalid.
-10034	**noSuchMessageError**	No configured message matches the one you tried to delete.
-10080	**badGainError**	The gain is invalid.
-10081	**badPretrigCountError**	The pretrigger sample count is invalid.
-10082	**badPosttrigCountError**	The posttrigger sample count is invalid.
-10083	**badTrigModeError**	The trigger mode is invalid.
-10084	**badTrigCountError**	The trigger count is invalid.
-10085	**badTrigRangeError**	The trigger range or trigger hysteresis window is invalid.
-10086	**badExtRefError**	The external reference is invalid.
-10087	**badTrigTypeError**	The trigger type is invalid.
-10088	**badTrigLevelError**	The trigger level is invalid.
-10089	**badTotalCountError**	The total count is inconsistent with the buffer size and pretrigger scan count or with the board type.

Table A-1. Status Code Summary (Continued)

Status Code	Status Name	Description
-10090	**badRPGError**	The individual range, polarity, and gain settings are valid but the combination is not allowed.
-10091	**badIterationsError**	You have attempted to use an invalid setting for the iterations parameter. The iterations value must be 0 or greater. Your device might be limited to only two values, 0 and 1.
-10092	**lowScanIntervalError**	Some devices require a time gap between the last sample in a scan and the start of the next scan. The scan interval you have specified does not provide a large enough gap for the board. See the SCAN_Start function in the language interface API for an explanation.
-10093	**fifoModeError**	FIFO mode waveform generation cannot be used because at least one condition is not satisfied.
-10100	**badPortWidthError**	The requested digital port width is not a multiple of the hardware port width or is not attainable by the DAQ hardware.
-10120	**gpctrBadApplicationError**	Invalid application used.
-10121	**gpctrBadCtrNumberError**	Invalid counterNumber used.
-10122	**gpctrBadParamValueError**	Invalid paramValue used.
-10123	**gpctrBadParamIDError**	Invalid paramID used.
-10124	**gpctrBadEntityIDError**	Invalid entityID used.
-10125	**gpctrBadActionError**	Invalid action used.
-10200	**EEPROMreadError**	Unable to read data from EEPROM.
-10201	**EEPROMwriteError**	Unable to write data to EEPROM.

Table A-1. Status Code Summary (Continued)

Status Code	Status Name	Description
-10240	**noDriverError**	The driver interface could not locate or open the driver.
-10241	**oldDriverError**	One of the driver files or the configuration utility is out of date.
-10242	**functionNotFoundError**	The specified function is not located in the driver.
-10243	**DSPconfigFileError**	The driver could not locate or open the configuration file, or the format of the configuration file is not compatible with the currently installed driver.
-10244	**deviceInitError**	The driver encountered a hardware-initialization error while attempting to configure the specified device.
-10245	**osInitError**	The driver encountered an operating-system error while attempting to perform an operation, or the operating system does not support an operation performed by the driver.
-10246	**communicationsError**	The driver is unable to communicate with the specified external device.
-10247	**DSPcmosConfigError**	The CMOS configuration-memory for the device is empty or invalid, or the configuration specified does not agree with the current configuration of the device, or the EISA system configuration is invalid.
-10248	**dupAddressError**	The base addresses for two or more devices are the same; consequently, the driver is unable to access the specified device.

Table A-1. Status Code Summary (Continued)

Status Code	Status Name	Description
-10249	**intConfigError**	The interrupt configuration is incorrect given the capabilities of the computer or device.
-10250	**dupIntError**	The interrupt levels for two or more devices are the same.
-10251	**dmaConfigError**	The DMA configuration is incorrect given the capabilities of the computer/DMA controller or device.
-10252	**dupDMAError**	The DMA channels for two or more devices are the same.
-10253	**jumperlessBoardError**	Unable to find one or more jumperless boards you have configured using the NI-DAQ Configuration Utility.
-10254	**DAQCardConfError**	Cannot configure the DAQCard because 1) the correct version of the card and socket services software is not installed; 2) the card in the PCMCIA socket is not a DAQCard; or 3) the base address and/or interrupt level requested are not available according to the card and socket services resource manager. Try different settings or use AutoAssign in the NI-DAQ configuration utility.
-10255	**remoteChassisDriverInitError**	There was an error in initializing the driver for Remote SCXI.
-10256	**comPortOpenError**	There was an error in opening the specified COM port.
-10257	**baseAddressError**	Bad base address specified in the configuration utility.

Table A-1. Status Code Summary (Continued)

Status Code	Status Name	Description
-10258	**dmaChannel1Error**	Bad DMA channel 1 specified in the configuration utility or by the operating system.
-10259	**dmaChannel2Error**	Bad DMA channel 2 specified in the configuration utility or by the operating system.
-10260	**dmaChannel3Error**	Bad DMA channel 3 specified in the configuration utility or by the operating system.
-10340	**noConnectError**	No RTSI signal/line is connected, or the specified signal and the specified line are not connected.
-10341	**badConnectError**	The RTSI signal/line cannot be connected as specified.
-10342	**multConnectError**	The specified RTSI signal is already being driven by a RTSI line, or the specified RTSI line is already being driven by a RTSI signal.
-10343	**SCXIConfigError**	The specified SCXI configuration parameters are invalid, or the function cannot be executed with the current SCXI configuration.
-10344	**chassisSynchedError**	The Remote SCXI unit is not synchronized with the host. Reset the chassis again to resynchronize it with the host.
-10345	**chassisMemAllocError**	The required amount of memory cannot be allocated on the Remote SCXI unit for the specified operation.

Table A-1. Status Code Summary (Continued)

Status Code	Status Name	Description
-10346	**badPacketError**	The packet received by the Remote SCXI unit is invalid. Check your serial port cable connections.
-10347	**chassisCommunicationError**	There was an error in sending a packet to the remote chassis. Check your serial port cable connections.
-10348	**waitingForReprogError**	The Remote SCXI unit is in reprogramming mode and is waiting for reprogramming commands from the host (NI-DAQ Configuration Utility).
-10349	**SCXIModuleTypeConflictError**	The module ID read from the SCXI module conflicts with the configured module type.
-10360	**DSPInitError**	The DSP driver was unable to load the kernel for its operating system.
-10370	**badScanListError**	The scan list is invalid; for example, you are mixing AMUX-64T channels and onboard channels, scanning SCXI channels out of order, or have specified a different starting channel for the same SCXI module. Also, the driver attempts to achieve complicated gain distributions over SCXI channels on the same module by manipulating the scan list and returns this error if it fails.
-10400	**userOwnedRsrcError**	The specified resource is owned by the user and cannot be accessed or modified by the driver.
-10401	**DSPunknownDeviceError**	The specified device is not a National Instruments product, or the driver does not support the device (e.g., the driver was released before the device was supported).

Table A-1. Status Code Summary (Continued)

Status Code	Status Name	Description
-10402	**deviceNotFoundError**	No device is located in the specified slot or at the specified address.
-10403	**DSPdeviceSupportError**	The specified device does not support the requested action (the driver recognizes the device, but the action is inappropriate for the device).
-10404	**noLineAvailError**	No line is available.
-10405	**noChanAvailError**	No channel is available.
-10406	**noGroupAvailError**	No group is available.
-10407	**lineBusyError**	The specified line is in use.
-10408	**chanBusyError**	The specified channel is in use.
-10409	**groupBusyError**	The specified group is in use.
-10410	**relatedLCGBusyError**	A related line, channel, or group is in use; if the driver configures the specified line, channel, or group, the configuration, data, or handshaking lines for the related line, channel, or group will be disturbed.
-10411	**counterBusyError**	The specified counter is in use.
-10412	**noGroupAssignError**	No group is assigned, or the specified line or channel cannot be assigned to a group.
-10413	**groupAssignError**	A group is already assigned, or the specified line or channel is already assigned to a group.
-10414	**reservedPinError**	The selected signal requires a pin that is reserved and configured only by NI-DAQ. You cannot configure this pin yourself.

Table A-1. Status Code Summary (Continued)

Status Code	Status Name	Description
-10415	**externalMuxSupporError**	This function does not support this device when an external multiplexer (such as an AMUX-64T or SCXI) is connected to it.
-10416	**DSPDataPathBusyError**	Either DAQ or WFM can use a PC memory buffer, but not both at the same time.
-10417	**SCXIModuleNotSupportedError**	At least one of the SCXI modules specified is not supported for the operation.
-10440	**sysOwnedRsrcError**	The specified resource is owned by the driver and cannot be accessed or modified by the user.
-10441	**memConfigError**	No memory is configured to support the current data-transfer mode, or the configured memory does not support the current data-transfer mode. (If block transfers are in use, the memory must be capable of performing block transfers.)
-10442	**memDisabledError**	The specified memory is disabled or is unavailable given the current addressing mode.
-10443	**memAlignmentError**	The transfer buffer is not aligned properly for the current data-transfer mode. E.g., the buffer is at an odd address, is not aligned to a 32-bit boundary, is not aligned to a 512-bit boundary, etc. Alternatively, the driver is unable to align the buffer because the buffer is too small.

Table A-1. Status Code Summary (Continued)

Status Code	Status Name	Description
-10444	**DSPmemFullError**	No more system memory is available on the heap, or no more memory is available on the device, or insufficient disk space is available.
-10445	**memLockError**	The transfer buffer cannot be locked into physical memory. On PC AT machines, portions of the DMA data acquisition buffer may be in an invalid DMA region, for example, above 16 megabytes.
-10446	**memPageError**	The transfer buffer contains a page break; system resources may require reprogramming when the page break is encountered.
-10447	**memPageLockError**	The operating environment is unable to grant a page lock.
-10448	**stackMemError**	The driver is unable to continue parsing a string input due to stack limitations.
-10449	**cacheMemError**	A cache-related error occurred, or caching is not supported in the current mode.
-10450	**physicalMemError**	A hardware error occurred in physical memory, or no memory is located at the specified address.
-10451	**virtualMemError**	The driver is unable to make the transfer buffer contiguous in virtual memory and therefore cannot lock it into physical memory; thus, the buffer cannot be used for DMA transfers.
-10452	**noIntAvailError**	No interrupt level is available for use.

Table A-1. Status Code Summary (Continued)

Status Code	Status Name	Description
-10453	**intInUseError**	The specified interrupt level is already in use by another device.
-10454	**noDMACError**	No DMA controller is available in the system.
-10455	**noDMAAvailError**	No DMA channel is available for use.
-10456	**DMAInUseError**	The specified DMA channel is already in use by another device.
-10457	**badDMAGroupError**	DMA cannot be configured for the specified group because it is too small, too large, or misaligned. Consult the device user manual to determine group ramifications with respect to DMA.
-10458	**diskFullError**	The storage disk you specified is full.
-10459	**DSPDLLInterfaceError**	The DLL could not be called due to an interface error.
-10460	**interfaceInteractionError**	You have mixed VIs from the DAQ library and the _DAQ compatibility library (LabVIEW 2.2 style VIs). You may switch between the two libraries only by running the DAQ VI Device Reset before calling _DAQ compatibility VIs or by running the compatibility VI Board Reset before calling DAQ VIs.
-10480	**muxMemFullError**	The scan list is too large to fit into the mux-gain memory of the board.
-10481	**bufferNotInterleavedError**	You cannot use DMA to transfer data from two buffers. You may be able to use interrupts.
-10560	**invalidDSPhandleError**	The DSP handle input is not valid .

Table A-1. Status Code Summary (Continued)

Status Code	Status Name	Description
-10600	**noSetupError**	No setup operation has been performed for the specified resources. Or, some resources require a specific ordering of calls for proper setup.
-10601	**multSetupError**	The specified resources have already been configured by a setup operation.
-10602	**noWriteError**	No output data has been written into the transfer buffer.
-10603	**groupWriteError**	The output data associated with a group must be for a single channel or must be for consecutive channels.
-10604	**activeWriteError**	Once data generation has started, only the transfer buffers originally written to may be updated. If DMA is active and a single transfer buffer contains interleaved channel-data, new data must be provided for all output channels currently using the DMA channel.
-10605	**endWriteError**	No data was written to the transfer buffer because the final data block has already been loaded.
-10606	**notArmedError**	The specified resource is not armed.
-10607	**armedError**	The specified resource is already armed.
-10608	**noTransferInProgError**	No transfer is in progress for the specified resource.
-10609	**transferInProgError**	A transfer is already in progress for the specified resource.
-10610	**transferPauseError**	A single output channel in a group may not be paused if the output data for the group is interleaved.

Table A-1. Status Code Summary (Continued)

Status Code	Status Name	Description
-10611	**badDirOnSomeLinesError**	Some of the lines in the specified channel are not configured for the transfer direction specified. For a write transfer, some lines are configured for input. For a read transfer, some lines are configured for output.
-10612	**badLineDirError**	The specified line does not support the specified transfer direction.
-10613	**badChanDirError**	The specified channel does not support the specified transfer direction.
-10614	**badGroupDirError**	The specified group does not support the specified transfer direction.
-10615	**masterClkError**	The clock configuration for the clock master is invalid.
-10616	**slaveClkError**	The clock configuration for the clock slave is invalid.
-10617	**noClkSrcError**	No source signal has been assigned to the clock resource.
-10618	**badClkSrcError**	The specified source signal cannot be assigned to the clock resource.
-10619	**multClkSrcError**	A source signal has already been assigned to the clock resource.
-10620	**noTrigError**	No trigger signal has been assigned to the trigger resource.
-10621	**badTrigError**	The specified trigger signal cannot be assigned to the trigger resource.
-10622	**preTrigError**	The pretrigger mode is not supported or is not available in the current configuration, or no pretrigger source has been assigned.

Table A-1. Status Code Summary (Continued)

Status Code	Status Name	Description
-10623	**postTrigError**	No posttrigger source has been assigned.
-10624	**delayTrigError**	The delayed trigger mode is not supported or is not available in the current configuration, or no delay source has been assigned.
-10625	**masterTrigError**	The trigger configuration for the trigger master is invalid.
-10626	**slaveTrigError**	The trigger configuration for the trigger slave is invalid.
-10627	**noTrigDrvError**	No signal has been assigned to the trigger resource.
-10628	**multTrigDrvError**	A signal has already been assigned to the trigger resource.
-10629	**invalidOpModeError**	The specified operating mode is invalid, or the resources have not been configured for the specified operating mode.
-10630	**invalidReadError**	The parameters specified to read data were invalid in the context of the acquisition. For example, an attempt was made to read 0 bytes from the transfer buffer, or an attempt was made to read past the end of the transfer buffer.
-10631	**noInfiniteModeError**	Continuous input or output transfers are not allowed in the current operating mode.
-10632	**someInputsIgnoredError**	Certain inputs were ignored because they are not relevant in the current operating mode.

Table A-1. Status Code Summary (Continued)

Status Code	Status Name	Description
-10633	**invalidRegenModeError**	The specified analog output regeneration mode is not allowed for this board.
-10634	**noContTransferInProgressError**	No continuous (double buffered) transfer is in progress for the specified resource.
-10635	**invalidSCXIOpModeError**	Either the SCXI operating mode specified in a configuration call is invalid, or a module is in the wrong operating mode to execute the function call.
-10636	**noContWithSynchError**	You cannot start a continuous (double-buffered) operation with a synchronous function call.
-10637	**bufferAlreadyConfigError**	Attempted to configure a buffer after the buffer had already been configured. You can configure a buffer only once.
-10680	**badChanGainError**	All channels of this board must have the same gain.
-10681	**badChanRangeError**	All channels of this board must have the same range.
-10682	**badChanPolarityError**	All channels of this board must be the same polarity.
-10683	**badChanCouplingError**	All channels of this board must have the same coupling.
-10684	**badChanInputModeError**	All channels of this board must have the same input mode.
-10685	**clkExceedsBrdsMaxConvRateError**	The clock rate exceeds the board's recommended maximum rate.

Table A-1. Status Code Summary (Continued)

Status Code	Status Name	Description
-10686	**scanListInvalidError**	A configuration change has invalidated the scan list.
-10687	**bufferInvalidError**	A configuration change has invalidated the acquisition buffer, or an acquisition buffer has not been configured.
-10688	**noTrigEnabledError**	The number of total scans and pretrigger scans implies that a triggered start is intended, but triggering is not enabled.
-10689	**digitalTrigBError**	Digital trigger B is illegal for the number of total scans and pretrigger scans specified.
-10690	**digitalTrigAandBError**	This board does not allow digital triggers A and B to be enabled at the same time.
-10691	**extConvRestrictionError**	This board does not allow an external sample clock with an external scan clock, start trigger, or stop trigger.
-10692	**chanClockDisabledError**	The acquisition cannot be started because the channel clock is disabled.
-10693	**extScanClockError**	You cannot use an external scan clock when doing a single scan of a single channel.
-10694	**unsafeSamplingFreqError**	The sample frequency exceeds the safe maximum rate for the hardware, gains, and filters used.
-10695	**DMAnotAllowedError**	You have set up an operation that requires the use of interrupts. DMA is not allowed. For example, some DAQ events, such as messaging and LabVIEW occurrences, require interrupts.

Table A-1. Status Code Summary (Continued)

Status Code	Status Name	Description
-10696	**multiRateModeError**	Multi-rate scanning cannot be used with the AMUX-64, SCXI, or pretriggered acquisitions.
-10697	**rateNotSupportedError**	Unable to convert your timebase/interval pair to match the actual hardware capabilities of this board.
-10698	**timebaseConflictError**	You cannot use this combination of scan and sample clock timebases for this board.
-10699	**polarityConflictError**	You cannot use this combination of scan and sample clock source polarities for this operation and board.
-10700	**signalConflictError**	You cannot use this combination of scan and convert clock signal sources for this operation and board.
-10701	**noLaterUpdateError**	The call had no effect because the specified channel had not been set for later internal update.
-10702	**prePostTriggerError**	Pretriggering and posttriggering cannot be used simultaneously on the Lab and 1200 series devices.
-10710	**noHandshakeModeError**	The specified port has not been configured for handshaking.
-10720	**noEventCtrError**	The specified counter is not configured for event-counting operation.
-10740	**SCXITrackHoldError**	A signal has already been assigned to the SCXI track-and-hold trigger line, or a control call was inappropriate because the specified module is not configured for one-channel operation.

Table A-1. Status Code Summary (Continued)

Status Code	Status Name	Description
-10780	**sc2040InputModeError**	When you have an SC2040 attached to your device, all analog input channels must be configured for differential input mode.
-10781	**outputTypeMustBeVoltageError**	The polarity of the output channel cannot be bipolar when outputting currents.
-10782	**sc2040HoldModeError**	The specified operation cannot be performed with the SC-2040 configured in hold mode.
-10783	**calConstPolarityConflictError**	Calibration constants in the load area have a different polarity from the current configuration. Therefore, you should load constants from factory.
-10800	**timeOutError**	The operation could not complete within the time limit.
-10801	**calibrationError**	An error occurred during the calibration process.
-10802	**dataNotAvailError**	The requested amount of data has not yet been acquired.
-10803	**transferStoppedError**	The transfer has been stopped to prevent regeneration of output data.
-10804	**earlyStopError**	The transfer stopped prior to reaching the end of the transfer buffer.

Table A-1. Status Code Summary (Continued)

Status Code	Status Name	Description
-10805	**overRunError**	The clock source for the input task is faster than the maximum clock rate the device supports. If you are allowing the driver to calculate the analog input channel clock rate, the driver bases the clock rate on the board type; so you should check that your board type is correct in the configuration utility.
-10806	**noTrigFoundError**	No trigger value was found in the input transfer buffer.
-10807	**earlyTrigError**	The trigger occurred before sufficient pretrigger data was acquired.
-10808	**LPTCommunicationError**	An error occurred in the parallel port communication with the DAQ device.
-10809	**gateSignalError**	Attempted to start a pulse width measurement with the pulse in the phase to be measured (e.g., high phase for high-level gating).
-10810	**internalDriverError**	An unexpected error occurred inside the driver when performing this given operation.
-10811	**internalKernelError**	An unexpected error occurred inside the kernel of the device while performing this operation.
-10840	**softwareError**	The contents or the location of the driver file was changed between accesses to the driver.
-10841	**firmwareError**	The firmware does not support the specified operation, or the firmware operation could not complete due to a data-integrity problem.

Table A-1. Status Code Summary (Continued)

Status Code	Status Name	Description
-10842	**hardwareError**	The hardware is not responding to the specified operation, or the response from the hardware is not consistent with the functionality of the hardware.
-10843	**underFlowError**	Because of system limitations, the driver could not write data to the device fast enough to keep up with the device throughput.
-10844	**underWriteError**	New data was not written to the output transfer buffer before the driver attempted to transfer the data to the device.
-10845	**overFlowError**	Because of system limitations, the driver could not read data from the device fast enough to keep up with the device throughput; the onboard device memory reported an overflow error.
-10846	**overWriteError**	The driver wrote new data into the input transfer buffer before the previously acquired data was read.
-10847	**dmaChainingError**	New buffer information was not available at the time of the DMA chaining interrupt; DMA transfers will terminate at the end of the currently active transfer buffer.
-10848	**noDMACountAvailError**	The driver could not obtain a valid reading from the transfer-count register in the DMA controller.
-10849	**OpenFileError**	The configuration file or DSP kernel file could not be opened.
-10850	**closeFileError**	Unable to close a file.

Table A-1. Status Code Summary (Continued)

Status Code	Status Name	Description
-10851	**fileSeekError**	Unable to seek within a file.
-10852	**readFileError**	Unable to read from a file.
-10853	**writeFileError**	Unable to write to a file.
-10854	**miscFileError**	An error occurred accessing a file.
-10855	**osUnsupportedError**	NI-DAQ does not support the current operation on this particular version of the operating system.
-10856	**osError**	An unexpected error occurred from the operating system while performing the given operation.
-10880	**updateRateChangeError**	A change to the update rate is not possible at this time because 1) when waveform generation is in progress, you cannot change the interval timebase or 2) when you make several changes in a row, you must give each change enough time to take effect before requesting further changes.
-10881	**partialTransferCompleteError**	You cannot do another transfer after a successful partial transfer.
-10882	**daqPollDataLossError**	The data collected on the Remote SCXI unit was overwritten before it could be transferred to the buffer in the host. Try using a slower data acquisition rate if possible.
-10883	**wfmPollDataLossError**	New data could not be transferred to the waveform buffer of the Remote SCXI unit to keep up with the waveform update rate. Try using a slower waveform update rate if possible.

Table A-1. Status Code Summary (Continued)

Status Code	Status Name	Description
-10884	**pretrigReorderError**	Could not rearrange data after a pretrigger acquisition completed.
-10920	**gpctrDataLossError**	One or more data points may have been lost during buffered GPCTR operations due to speed limitations of your system.
-10940	**chassisResponseTimeoutError**	No response was received from the Remote SCXI unit within the specified time limit.
-10941	**reprogrammingFailedError**	Reprogramming the Remote SCXI unit was unsuccessful. Please try again.
-10942	**invalidResetSignatureError**	An invalid reset signature was sent from the host to the Remote SCXI unit.

Analog Input Channel and Gain Settings and Voltage Calculation

Appendix
B

This appendix lists the valid channel and gain settings for DAQ boards, describes how NI-DAQ calculates voltage, and describes the measurement of offset and gain adjustment.

DAQ Device Analog Input Channel Settings

Table B-1 lists the valid analog input (ADC) channel settings. If you have one or more AMUX-64T boards and an MIO board, see Chapter 13, *AMUX-64T External Multiplexer Device*, in the *DAQ Hardware Overview Guide* for more information.

Table B-1. Valid Analog Input Channel Settings

Device	Settings	
	Single-ended Configuration	**Differential Configuration**
MIO and AI devices (except AT-MIO-64E-3, AT-MIO-64F-5, and DAQPad-MIO-16XE-50)	0–15	0–7
AT-MIO-64F-5	0–63	0–7 and 16–39
AT-MIO-64E-3	0–63	0–7, 16–23, 32–39, 48–55
Lab and 1200 Series devices	0–7	0, 2, 4, 6
LPM devices	0–15	N/A
DAQCard-700	0–15	0–7
516 devices, DAQCard-500	0–7	0–3 (516 devices only)

Table B-1. Valid Analog Input Channel Settings

Device	Settings	
	Single-ended Configuration	**Differential Configuration**
DAQPad-MIO-16XE-50	0–15 and `ND_CJ_TEMP`†	0–7 and `ND_CJ_TEMP`†

†`ND_CJ_TEMP` is a constant that is defined in the following header files:

- C programmers—`NIDAQCNS.H` (`DATAACQ.H` for LabWindows/CVI)
- BASIC programmers—`NIDAQCNS.INC` (Visual Basic for Windows programmers should refer to the *Programming Language Considerations* section in Chapter 1, *Using the NI-DAQ Functions,* for more information.)
- Pascal programmers—`NIDAQCNS.PAS`

DAQ Device Gain Settings

Table B-2 lists the valid gain settings for DAQ devices.

Table B-2. Valid Gain Settings

Device	Valid Gain Settings
AT-MIO-16L, AT-MIO-16DL	1, 10, 100, 500
AT-MIO-16H, AT-MIO-16DH	1, 2, 4, 8
AT-MIO-16F-5, AT-MIO-64F-5, and most E Series devices	-1 (for a gain of 0.5), 1, 2, 5, 10, 20, 50, 100
AT-MIO-16XE-50, DAQCard-AI-16XE-50, DAQPad-MIO-16XE-50, NEC-MIO-16XE-50, NEC-AI-16XE-50	1, 2, 10, 100
AT-MIO-16X, Lab and 1200 Series devices	1, 2, 5, 10, 20, 50, 100
DAQCard-500/700, 516 and LPM devices	gain is ignored because gain is always 1

Voltage Calculation

`AI_VScale` and `DAQ_VScale` calculate voltage from **reading** as follows:

$$\text{voltage} = \left(\frac{\textbf{reading} - \textbf{offset}}{\textbf{maxReading}}\right) \times \left(\frac{\textbf{maxVolt}}{\textbf{gain} \times \textbf{gainAdjust}}\right)$$

where:

- **maxReading** is the maximum binary reading for the given board, channel, range, and polarity.
- **maxVolt** is the maximum voltage the board can measure at a gain of 1 in the given range and polarity.

Table B-3 lists the values of **maxReading** and **maxVolt** for different boards.

Table B-3. The Values of maxReading and maxVolt

Device	**Unipolar Mode**		**Bipolar Mode**	
	maxReading	**maxVolt**	**maxReading**	**maxVolt**
MIO-16, AT-MIO-16D	4,096	*	2,048	*
AT-MIO-16F-5, AT-MIO-64F-5, and most E Series devices	4,096	10 V	2,048	5 V
16-bit E Series devices and AT-MIO-16X	65,536	10 V	32,768	10 V
Lab-PC+, Lab-PC-1200, Lab-PC-1200AI, DAQPad-1200, DAQCard-1200	4,096	10 V	2,048	5 V
DAQCard-700, LPM devices	4,096	*	2,048	*
516 devices	N/A	N/A	32,768	5 V
DAQCard-500	N/A	N/A	2,048	5 V
* The value of **maxVolt** depends on **inputRange**, as discussed in `AI_Configure`.				

For the PC-LPM-16 and DAQCard-1200, gain is ignored, and the following formula is used:

$$\text{voltage} = \left(\frac{\mathbf{reading} - \mathbf{offset}}{\mathbf{maxReading}}\right) \times (\mathbf{maxVolt})$$

Offset and Gain Adjustment

Measurement of Offset

To determine the **offset** parameter used in the `AI_VScale` and `DAQ_VScale` functions, follow this procedure:

1. Ground analog input channel *i*, where *i* can be any valid input channel.
2. Call the `AI_Read` function with **gain** set to the gain that will be used in your real acquisition (*g*). The reading given by the `AI_Read` function is the value of **offset**. The offset is only valid for the gain setting at which it was measured. Remember that the data type of **offset** in the `AI_VScale` and `DAQ_VScale` functions is floating point, so if you use `AI_Read` to get the offset, you will have to typecast it before passing it to the scale function.

Note: ***Another way to read the offset is to perform multiple readings using a DAQ function call and average them to be more accurate and reduce the effects of noise.***

Measurement of Gain Adjustment

To determine the **gainAdjust** parameter used in the `AI_VScale` and `DAQ_VScale` functions, follow this procedure:

1. Connect the known voltage V_{in} to channel *i*.
2. Call the `AI_Read` function with gain equal to *g*. Use the reading returned by `AI_Read` with the offset value determined above to calculate the real gain.

Note: ***You can use the DAQ functions to take many readings and average them instead of using the `AI_Read` function.***

The real gain is computed as follows:

$$G_R = \left(\frac{\textbf{reading} - \textbf{offset}}{\textbf{maxReading}}\right) \times \left(\frac{\textbf{maxVolt}}{V_{in}}\right)$$

The gain adjustment is computed as follows:

$$\textbf{gainAdjust} = \left[1 - \frac{(g - G_R)}{g}\right]$$

NI-DAQ Function Support

Appendix
C

This appendix contains tables that show which DAQ hardware each NI-DAQ function call supports.

The NI-DAQ functions are listed in alphabetical order. A check mark indicates the hardware that the function supports. If you attempt to call an NI-DAQ function using a device that the function does not support, NI-DAQ returns a **deviceSupportError**.

Table C-1 lists the NI-DAQ functions for plug-in DAQ boards and cards and SCXI DAQ and control modules. Table C-2 lists the SCXI functions used with SCXI modules and compatible DAQ boards.

Table C-1. NI-DAQ Functions

Device \ Function	A2000_Calibrate	A2000_Config	A2150_Calibrate	AI_Check	AI_Clear	AI_Configure	AI_Mux_Config	AI_Read	AI_Read_Scan	AI_Setup	AI_VRead	AI_VRead_Scan	AI_VScale	Align_DMA_Buffer	AO_Calibrate	AO_Change_Parameter	AO_Configure
SCXI-1200				✓	✓	✓		✓		✓	✓		✓				✓
PC-TIO-10																	
PC-OPDIO-16																	
PC-DIO-96/PnP																	
PC-DIO-24																	
MIO E Series				✓	✓	✓	✓	✓	✓	✓	✓	✓	✓			✓	✓
Lab-PC+/Lab-PC-1200/AI				✓	✓	✓		✓		✓	✓		✓				✓
EISA-A2000	✓	✓															
DAQPad-1200				✓	✓	✓		✓		✓	✓		✓				✓
DAQCard DIO-24																	
DAQCard-1200				✓	✓	✓		✓		✓	✓		✓				✓
DAQCard-500/700				✓	✓	✓		✓		✓	✓		✓				
AT-MIO-64F-5				✓	✓	✓	✓	✓		✓	✓		✓				✓
AT-MIO-16X				✓	✓	✓	✓	✓		✓	✓		✓			✓	✓
AT-MIO-16F-5				✓	✓	✓	✓	✓		✓	✓		✓	✓			✓
AT-MIO-16D				✓	✓	✓	✓	✓		✓	✓		✓				✓
AT-MIO-16				✓	✓	✓	✓	✓		✓	✓		✓				✓
AT-DSP2200																	
AT-DIO-32F														✓			
AT-AO-6/10															✓		✓
AT-A2150			✓														
AO-2DC																✓	✓
AI E Series				✓	✓	✓	✓	✓	✓	✓	✓	✓	✓				
516 and LPM Devices				✓	✓	✓		✓		✓	✓		✓				

Table C-1. NI-DAQ Functions (Continued)

Function	Device																							
	516 and LPM Devices	AI E Series	AO-2DC	AT-A2150	AT-AO-6/10	AT-DIO-32F	AT-DSP2200	AT-MIO-16	AT-MIO-16D	AT-MIO-16F-5	AT-MIO-16X	AT-MIO-64F-5	DAQCard-500/700	DAQCard-1200	DAQCard DIO-24	DAQPad-1200	EISA-A2000	Lab-PC+/Lab-PC-1200/AI	MIO E Series	PC-DIO-24	PC-DIO-96/PnP	PC-OPDIO-16	PC-TIO-10	SCXI-1200
AO_Update					√			√	√	√	√	√		√		√		√	√					√
AO_VScale					√		√	√	√	√	√	√		√		√		√	√					√
AO_VWrite			√		√		√	√	√	√	√	√		√		√		√	√					√
AO_Write					√		√	√	√	√	√	√		√		√		√	√					√
Calibrate_1200																√								√
Calibrate_E_Series		√																	√					
Config_Alarm_Deadband	√	††						√	√	√	√	√	√	√		√		√	√					√
Config_ATrig_Event_Message	√	††						√	√	√	√	√	√	√		√		√	√					√
Config_DAQ_Event_Message	√	††				√		√	√	√	√	√	√	√	√	√		√	√	√	√		√	√
Configure_HW_Analog_Trigger		†																	†					
CTR_Config								√	√	√	√	√					√						√	
CTR_EvCount								√	√	√	√	√					√						√	
CTR_EvRead								√	√	√	√	√					√						√	
CTR_FOUT_Config								√	√	√	√	√											√	
CTR_Period								√	√	√	√	√					√						√	
CTR_Pulse								√	√	√	√	√					√						√	
CTR_Rate								√	√	√	√	√					√						√	

Table C-1. NI-DAQ Functions (Continued)

Device \ Function	CTR_Reset	CTR_Restart	CTR_Simul_Op	CTR_Square	CTR_State	CTR_Stop	DAQ_Check	DAQ_Clear	DAQ_Config	DAQ_DB_Config	DAQ_DB_HalfReady	DAQ_DB_StrTransfer	DAQ_DB_Transfer	DAQ_Monitor	DAQ_Op	DAQ_Rate	DAQ_Start
SCXI-1200							✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
PC-TIO-10	✓	✓	✓	✓	✓	✓											
PC-OPDIO-16																	
PC-DIO-96/PnP																	
PC-DIO-24																	
MIO E Series							✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Lab-PC+/Lab-PC-1200/AI							✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
EISA-A2000	✓	✓		✓	✓	✓										✓	
DAQPad-1200							✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
DAQCard DIO-24																	
DAQCard-1200							✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
DAQCard-500/700							✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
AT-MIO-64F-5	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
AT-MIO-16X	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
AT-MIO-16F-5	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
AT-MIO-16D	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
AT-MIO-16	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
AT-DSP2200																	
AT-DIO-32F																	
AT-AO-6/10																	
AT-A2150																	
AO-2DC																	
AI E Series							✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
516 and LPM Devices							✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

Table C-1. NI-DAQ Functions (Continued)

Function	Device																							
	516 and LPM Devices	AI E Series	AO-2DC	AT-A2150	AT-AO-6/10	AT-DIO-32F	AT-DSP2200	AT-MIO-16	AT-MIO-16D	AT-MIO-16F-5	AT-MIO-16X	AT-MIO-64F-5	DAQCard-500/700	DAQCard-1200	DAQCard DIO-24	DAQPad-1200	EISA-A2000	Lab-PC+/Lab-PC-1200/AI	MIO E Series	PC-DIO-24	PC-DIO-96/PnP	PC-OPDIO-16	PC-TIO-10	SCXI-1200
`DAQ_StopTrigger_Config`		✓						✓	✓	✓	✓	✓		✓		✓		✓	✓					✓
`DAQ_to_Disk`	✓	✓						✓	✓	✓	✓	✓	✓	✓		✓	✓	✓	✓					✓
`DAQ_VScale`	✓	✓						✓	✓	✓	✓	✓	✓	✓		✓		✓	✓					✓
`DIG_Block_Check`		*				✓			✓					✓	✓	✓		✓	*	✓	✓			✓
`DIG_Block_Clear`		*				✓			✓					✓	✓	✓		✓	*	✓	✓			✓
`DIG_Block_In`		*				✓			✓					✓	✓	✓		✓	*	✓	✓			✓
`DIG_Block_Out`		*				✓			✓					✓	✓	✓		✓	*	✓	✓			✓
`DIG_Block_PG_Config`						✓																		
`DIG_DB_Config`						✓																		
`DIG_DB_HalfReady`						✓																		
`DIG_DB_StrTransfer`						✓																		
`DIG_DB_Transfer`						✓																		
`DIG_Grp_Config`						✓																		
`DIG_Grp_Mode`						✓																		
`DIG_Grp_Status`						✓																		
`DIG_In_Grp`						✓																		
`DIG_In_Line`	✓	✓	✓		✓	✓		✓	✓	✓	✓	✓	✓	✓	✓	✓		✓	✓	✓	✓	✓	✓	✓

Table C-1. NI-DAQ Functions (Continued)

Function	Device																							
	516 and LPM Devices	AI E Series	AO-2DC	AT-A2150	AT-AO-6/10	AT-DIO-32F	AT-DSP2200	AT-MIO-16	AT-MIO-16D	AT-MIO-16F-5	AT-MIO-16X	AT-MIO-64F-5	DAQCard-500/700	DAQCard-1200	DAQCard DIO-24	DAQPad-1200	EISA-A2000	Lab-PC+/Lab-PC-1200/AI	MIO E Series	PC-DIO-24	PC-DIO-96/PnP	PC-OPDIO-16	PC-TIO-10	SCXI-1200
`DIG_In_Port`	✓	✓	✓		✓	✓		✓	✓	✓	✓	✓	✓	✓	✓	✓		✓	✓	✓	✓	✓	✓	✓
`DIG_Line_Config`		✓																	✓				✓	
`DIG_Out_Grp`						✓																		
`DIG_Out_Line`	✓	✓	✓		✓	✓		✓	✓	✓	✓	✓	✓	✓	✓	✓		✓	✓	✓	✓	✓	✓	✓
`DIG_Out_Port`	✓	✓	✓		✓	✓		✓	✓	✓	✓	✓	✓	✓	✓	✓		✓	✓	✓	✓	✓	✓	✓
`DIG_Prt_Config`	✓	✓	✓		✓	✓		✓	✓	✓	✓	✓	✓	✓	✓	✓		✓	✓	✓	✓	✓	✓	✓
`DIG_Prt_Status`									✓					✓	✓	✓		✓		✓	✓			✓
`DIG_SCAN_Setup`		*							✓					✓	✓	✓		✓	*	✓	✓			✓
`DSP2200_Calibrate`							✓																	
`DSP2200_Config`							✓																	
`Get_DAQ_Event`	✓	✓				✓		✓	✓	✓	✓	✓	✓	✓	✓	✓		✓	✓	✓	✓		✓	✓
`Get_DAQ_Device_Info`	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
`Get_NI_DAQ_Version`	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
`GPCTR_Change_Parameter`		✓																	✓					
`GPCTR_Config_Buffer`		✓																	✓					
`GPCTR_Control`		✓																	✓					
`GPCTR_Set_Application`		✓																	✓					

Table C-1. NI-DAQ Functions (Continued)

Device \ Function	GPCTR_Watch	ICTR_Read	ICTR_Reset	ICTR_Setup	Init_DA_Brds	Lab_ISCAN_Check	Lab_ISCAN_Op	Lab_ISCAN_Start	Lab_ISCAN_to_Disk	LPM16_Calibrate	MAI_Arm	MAI_Clear	MAI_Coupling	MAI_Read	MAI_Scale	MAI_Setup	Master_Slave_Config
SCXI-1200		✓	✓	✓	✓	✓	✓	✓	✓								
PC-TIO-10					✓												
PC-OPDIO-16					✓												
PC-DIO-96/PnP					✓												
PC-DIO-24					✓												
MIO E Series	✓				✓												
Lab-PC+/Lab-PC-1200/AI		✓	✓	✓	✓	✓	✓	✓	✓								
EISA-A2000					✓						✓	✓	✓	✓	✓	✓	✓
DAQPad-1200		✓	✓	✓	✓	✓	✓	✓	✓								
DAQCard DIO-24					✓												
DAQCard-1200		✓	✓	✓	✓	✓	✓	✓	✓								
DAQCard-500/700		✓	✓	✓	✓	✓	✓	✓	✓								
AT-MIO-64F-5					✓												
AT-MIO-16X					✓												
AT-MIO-16F-5					✓												
AT-MIO-16D					✓												
AT-MIO-16					✓												
AT-DSP2200					✓							✓	✓	✓	✓	✓	✓
AT-DIO-32F					✓												
AT-AO-6/10					✓												
AT-A2150					✓							✓	✓	✓	✓	✓	✓
AO-2DC					✓												
AI E Series	✓				✓												
516 and LPM Devices		✓	✓	✓	✓	✓	✓	✓	✓	**							

Table C-1. NI-DAQ Functions (Continued)

Device \ Function	MDAQ_Check	MDAQ_Clear	MDAQ_Get	MDAQ_ScanRate	MDAQ_Setup	MDAQ_Start	MDAQ_Stop	MDAQ_StrGet	MDAQ_Trig_Delay	MDAQ_Trig_Select	MIO_Calibrate	MIO_Config	NI_DAQ_Mem_Alloc	NI_DAQ_Mem_Attributes	NI_DAQ_Mem_Copy	NI_DAQ_Mem_Free	NI_DAQ_Mem_Lock
SCXI-1200												✓	✓	✓	✓	✓	✓
PC-TIO-10													✓	✓	✓	✓	✓
PC-OPDIO-16																	
PC-DIO-96/PnP													✓	✓	✓	✓	✓
PC-DIO-24													✓	✓	✓	✓	✓
MIO E Series												✓	✓	✓	✓	✓	✓
Lab-PC+/Lab-PC-1200/AI													✓	✓	✓	✓	✓
EISA-A2000	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓			✓	✓	✓	✓	✓
DAQPad-1200												✓	✓	✓	✓	✓	✓
DAQCard DIO-24													✓	✓	✓	✓	✓
DAQCard-1200													✓	✓	✓	✓	✓
DAQCard-500/700													✓	✓	✓	✓	✓
AT-MIO-64F-5											✓	✓	✓	✓	✓	✓	✓
AT-MIO-16X											✓		✓	✓	✓	✓	✓
AT-MIO-16F-5											✓	✓	✓	✓	✓	✓	✓
AT-MIO-16D													✓	✓	✓	✓	✓
AT-MIO-16													✓	✓	✓	✓	✓
AT-DSP2200	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓			✓	✓	✓	✓	✓
AT-DIO-32F													✓	✓	✓	✓	✓
AT-AO-6/10													✓	✓	✓	✓	✓
AT-A2150	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓			✓	✓	✓	✓	✓
AO-2DC																	
AI E Series												✓	✓	✓	✓	✓	✓
516 and LPM Devices													✓	✓	✓	✓	✓

Table C-1. NI-DAQ Functions (Continued)

Device \ Function	NI_DAQ_Mem_Unlock	Peek_DAQ_Event	RTSI_Clear	RTSI_Clock	RTSI_Conn	RTSI_DisConn	SC_2040_Config	SCAN_Demux	SCAN_Op	SCAN_Sequence_Demux	SCAN_Sequence_Retrieve	SCAN_Sequence_Setup	SCAN_Setup	SCAN_Start	SCAN_to_Disk	Select_Signal	Set_DAQ_Device_Info
SCXI-1200	✓	✓						✓									✓
PC-TIO-10	✓	✓															
PC-OPDIO-16																	
PC-DIO-96/PnP	✓	✓															
PC-DIO-24	✓	✓															
MIO E Series	✓	✓					✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Lab-PC+/Lab-PC-1200/AI	✓	✓						✓									✓
EISA-A2000	✓		✓	✓	✓	✓		✓									✓
DAQPad-1200	✓	✓						✓									✓
DAQCard DIO-24	✓	✓															
DAQCard-1200	✓	✓						✓									✓
DAQCard-500/700	✓	✓						✓									✓
AT-MIO-64F-5	✓	✓	✓	✓	✓	✓		✓	✓	✓	✓	✓	✓	✓	✓		✓
AT-MIO-16X	✓	✓	✓	✓	✓	✓		✓	✓	✓	✓	✓	✓	✓	✓		✓
AT-MIO-16F-5	✓	✓	✓		✓	✓		✓	✓	✓	✓	✓	✓	✓	✓		✓
AT-MIO-16D	✓	✓	✓		✓	✓		✓	✓				✓	✓	✓		✓
AT-MIO-16	✓	✓	✓		✓	✓		✓	✓				✓	✓	✓		✓
AT-DSP2200	✓		✓	✓	✓	✓		✓									✓
AT-DIO-32F	✓	✓	✓		✓	✓											
AT-AO-6/10	✓		✓	✓	✓	✓											
AT-A2150	✓		✓	✓	✓	✓		✓									✓
AO-2DC																	
AI E Series	✓	✓					✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
516 and LPM Devices	✓	✓						✓									✓

Table C-1. NI-DAQ Functions (Continued)

Function	Device																							
	516 and LPM Devices	AI E Series	AO-2DC	AT-A2150	AT-AO-6/10	AT-DIO-32F	AT-DSP2200	AT-MIO-16	AT-MIO-16D	AT-MIO-16F-5	AT-MIO-16X	AT-MIO-64F-5	DAQCard-500/700	DAQCard-1200	DAQCard DIO-24	DAQPad-1200	EISA-A2000	Lab-PC+/Lab-PC-1200/AI	MIO E Series	PC-DIO-24	PC-DIO-96/PnP	PC-OPDIO-16	PC-TIO-10	SCXI-1200
`Timeout_Config`	√	√			√	√	√	√	√	√	√	√	√	√		√		√	√					√
`Trigger_Window_Config`				√			√																	
`WFM_Chan_Control`					√		√	√	√	√	√	√		√		√		√	√					√
`WFM_Check`					√		√	√	√	√	√	√		√		√		√	√					√
`WFM_ClockRate`					√		√	√	√	√	√	√		√		√		√	√					√
`WFM_DB_Config`					√		√	√	√	√	√	√		√		√		√	√					√
`WFM_DB_HalfReady`					√		√	√	√	√	√	√		√		√		√	√					√
`WFM_DB_StrTransfer`					√		√	√	√	√	√	√		√		√		√	√					√
`WFM_DB_Transfer`					√		√	√	√	√	√	√		√		√		√	√					√
`WFM_from_Disk`					√		√	√	√	√	√	√		√		√		√	√					√
`WFM_Group_Control`		√			√		√	√	√	√	√	√		√		√		√	√					√
`WFM_Group_Setup`		√			√		√	√	√	√	√	√		√		√		√	√					√
`WFM_Load`		√			√		√	√	√	√	√	√		√		√		√	√					√
`WFM_Op`		√			√		√	√	√	√	√	√		√		√		√	√					√
`WFM_Rate`		√			√			√	√	√	√	√		√		√		√	√					√
`WFM_Scale`		√			√		√	√	√	√	√	√		√		√		√	√					√

* AT-MIO-16DE-10 only ** LPM devices only † AT-MIO-16E-2, AT-MIO-64E-3, NEC-MIO-16E-4, NEC-AI-16E-4, and AT-MIO-16E-1 only
†† AT-AI-16XE-10, NEC-AI-16XE-50, and NEC-AI-16E-4 only

Table C-2. SCXI Function and Hardware Support

Function	Module												Device						
	SCXI-1100	SCXI-1102	SCXI-1120	SCXI-1121	SCXI-1122	SCXI-1124	SCXI-1140	SCXI-1141	SCXI-1160	SCXI-1161	SCXI-1162/1162HV	SCXI-1163/1163R	SCXI-1200	AO-2DC	DAQCard-700	DIO Devices	Lab-PC+	MIO and AI Devices	LPM Devices
`SCXI_AO_Write`						√													
`SCXI_Cal_Constants`	√	√	√	√	√	√	√	√											
`SCXI_Calibrate_Setup`	√	√		√	√			√											
`SCXI_Change_Chan`	√	√	√	√	√		√	√											
`SCXI_Configure_Filter`					√			√											
`SCXI_Get_Chassis_Info`	√	√	√	√	√	√	√	√	√	√	√	√							
`SCXI_Get_Module_Info`	√	√	√	√	√	√	√	√	√	√	√	√	√						
`SCXI_Get_State`									√	√	√	√							
`SCXI_Get_Status`		√				√			√										
`SCXI_Load_Config`	√	√	√	√	√	√	√	√	√	√	√	√	√		√	√	√	√	√
`SCXI_MuxCtr_Setup`													√				√	√	
`SCXI_Reset`	√	√	√	√	√	√	√	√	√	√	√	√	√						
`SCXI_Scale`	√	√	√	√	√		√	√					√		√		√	√	√
`SCXI_SCAN_Setup`	√	√	√	√	√		√	√					√				√	√	
`SCXI_Set_Config`	√	√	√	√	√	√	√	√	√	√	√	√			√	√	√	√	√
`SCXI_Set_Gain`	√				√			√											
`SCXI_Set_Input_Mode`					√														
`SCXI_Set_State`									√	√		√							
`SCXI_Single_Chan_Setup`	√	√	√	√	√		√	√					√		√		√	√	√
`SCXI_Track_Hold_Control`							√						√		√		√	√	√
`SCXI_Track_Hold_Setup`							√						√		√		√	√	√

Appendix D

Customer Communication

For your convenience, this appendix contains forms to help you gather the information necessary to help us solve your technical problems and a form you can use to comment on the product documentation. When you contact us, we need the information on the Technical Support Form and the configuration form, if your manual contains one, about your system configuration to answer your questions as quickly as possible.

National Instruments has technical assistance through electronic, fax, and telephone systems to quickly provide the information you need. Our electronic services include a bulletin board service, an FTP site, a FaxBack system, and e-mail support. If you have a hardware or software problem, first try the electronic support systems. If the information available on these systems does not answer your questions, we offer fax and telephone support through our technical support centers, which are staffed by applications engineers.

Electronic Services

Bulletin Board Support

National Instruments has BBS and FTP sites dedicated for 24-hour support with a collection of files and documents to answer most common customer questions. From these sites, you can also download the latest instrument drivers, updates, and example programs. For recorded instructions on how to use the bulletin board and FTP services and for BBS automated information, call (512) 795-6990. You can access these services at:

United States: (512) 794-5422 or (800) 327-3077
Up to 14,400 baud, 8 data bits, 1 stop bit, no parity

United Kingdom: 01635 551422
Up to 9,600 baud, 8 data bits, 1 stop bit, no parity

France: 1 48 65 15 59
Up to 9,600 baud, 8 data bits, 1 stop bit, no parity

FTP Support

To access our FTP site, log on to our Internet host, `ftp.natinst.com`, as anonymous and use your Internet address, such as `joesmith@anywhere.com`, as your password. The support files and documents are located in the `/support` directories.

FaxBack Support

FaxBack is a 24-hour information retrieval system containing a library of documents on a wide range of technical information. You can access FaxBack from a touch-tone telephone at (512) 418-1111

E-Mail Support (currently U.S. only)

You can submit technical support questions to the appropriate applications engineering team through e-mail at the Internet addresses listed below. Remember to include your name, address, and phone number so we can contact you with solutions and suggestions.

GPIB: `gpib.support@natinst.com`	LabVIEW: `lv.support@natinst.com`
DAQ: `daq.support@natinst.com`	HiQ: `hiq.support@natinst.com`
VXI: `vxi.support@natinst.com`	VISA: `visa.support@natinst.com`
LabWindows: `lw.support@natinst.com`	Lookout: `lookout.support@natinst.com`

Fax and Telephone Support

National Instruments has branch offices all over the world. Use the list below to find the technical support number for your country. If there is no National Instruments office in your country, contact the source from which you purchased your software to obtain support.

	Telephone	Fax
Australia	03 9 879 9422	03 9 879 9179
Austria	0662 45 79 90 0	0662 45 79 90 19
Belgium	02 757 00 20	02 757 03 11
Canada (Ontario)	519 622 9310	
Canada (Quebec)	514 694 8521	514 694 4399
Denmark	45 76 26 00	45 76 26 02
Finland	90 527 2321	90 502 2930
France	1 48 14 24 24	1 48 14 24 14
Germany	089 741 31 30	089 714 60 35
Hong Kong	2645 3186	2686 8505
Italy	02 413091	02 41309215
Japan	03 5472 2970	03 5472 2977
Korea	02 596 7456	02 596 7455
Mexico	95 800 010 0793	5 520 3282
Netherlands	0348 433466	0348 430673
Norway	32 84 84 00	32 84 86 00
Singapore	2265886	2265887
Spain	91 640 0085	91 640 0533
Sweden	08 730 49 70	08 730 43 70
Switzerland	056 200 51 51	056 200 51 55
Taiwan	02 377 1200	02 737 4644
U.K.	01635 523545	01635 523154

Technical Support Form

Photocopy this form and update it each time you make changes to your software or hardware, and use the completed copy of this form as a reference for your current configuration. Completing this form accurately before contacting National Instruments for technical support helps our applications engineers answer your questions more efficiently.

If you are using any National Instruments hardware or software products related to this problem, include the configuration forms from their user manuals. Include additional pages if necessary.

Name ____

Company ____

Address ____

____

Fax (___)____________________ Phone (___) ____

Computer brand _________________ Model _________________ Processor___________________

Operating system (include version number) ____

Clock speed ______MHz RAM _____MB Display adapter ____________________________

Mouse ___yes ___no Other adapters installed _______________________________________

Hard disk capacity _____MB Brand ____

Instruments used ____

____

National Instruments hardware product model ___________ Revision _________________________

Configuration ____

National Instruments software product ______________________________ Version _____________

Configuration ____

The problem is: ____

____

____

____

____

List any error messages: ____

____

____

The following steps reproduce the problem:____

____

____

____

____

____

NI-DAQ for PC Compatibles Hardware and Software Configuration Form

Record the settings and revisions of your hardware and software on the line to the right of each item. Complete a new copy of this form each time you revise your software or hardware configuration, and use this form as a reference for your current configuration. Completing this form accurately before contacting National Instruments for technical support helps our applications engineers answer your questions more efficiently.

National Instruments Products

DAQ hardware ______________________________

Interrupt level of hardware ______________________________

DMA channels of hardware ______________________________

Base I/O address of hardware ______________________________

Programming choice ______________________________

HiQ, NI-DAQ, LabVIEW, or LabWindows version ______________________________

Other boards in system ______________________________

Base I/O address of other boards ______________________________

DMA channels of other boards ______________________________

Interrupt level of other boards ______________________________

Other Products

Computer make and model ______________________________

Microprocessor ______________________________

Clock frequency or speed ______________________________

Type of video board installed ______________________________

Operating system version ______________________________

Operating system mode ______________________________

Programming language ______________________________

Programming language version ______________________________

Other boards in system ______________________________

Base I/O address of other boards ______________________________

DMA channels of other boards ______________________________

Interrupt level of other boards ______________________________

Documentation Comment Form

National Instruments encourages you to comment on the documentation supplied with our products. This information helps us provide quality products to meet your needs.

Title: NI-DAQ® Function Reference Manual for PC Compatibles

Edition Date: June 1996

Part Number: 320499D-01

Please comment on the completeness, clarity, and organization of the manual.

____

____

____

____

____

____

____

If you find errors in the manual, please record the page numbers and describe the errors.

____

____

____

____

____

____

____

Thank you for your help.

Name ____

Title ____

Company ____

Address ____

____

Phone (   ) ____

Mail to: Technical Publications
National Instruments Corporation
6504 Bridge Point Parkway
Austin, TX 78730-5039

Fax to: Technical Publications
National Instruments Corporation
(512) 794-5678

Glossary

Prefix	Meaning	Value
n-	nano-	10^{-9}
µ-	micro-	10^{-6}
m-	milli-	10^{-3}
k-	kilo-	10^{3}
M-	mega-	10^{6}

°	degree
≤	less than or equal to
-	minus
+	plus
%	percent
Ω	ohm
AC	alternating current
ACK	acknowledge
A/D	analog-to-digital
ADC	A/D converter
AI	Analog Input
AMUX	AMUX-64T
API	application programming interface
BCD	binary-coded decimal

C	Celsius
CI	computing index
DC	direct current
D/A	digital-to-analog
DAC	D/A converter
DAQ	data acquisition
DIG	digital
DIO	digital I/O
DLL	dynamic link library
DMA	direct memory access
DOS	Disk Operating System
DSP	digital signal processing
EEPROM	electronically erasable programmable read-only memory
EISA	Extended Industry Standard Architecture
FIFO	first-in-first-out
Hz	hertz
ID	identification
IEEE	Institute of Electrical and Electronics Engineers
I/O	input/output
ISA	Industry Standard Architecture
Kword	1,024 words of memory
LSB	least significant bit
MB	megabytes of memory
MC	Micro Channel
min	minutes
MIO	multifunction I/O

NC	Normally Closed
NO	Normally Open
OUT	Output
PC	personal computer
pts	points
REQ	request
rms	root mean square
RTSI	Real-Time System Integration
SCXI	Signal Conditioning eXtensions for Instrumentation
SDK	Software Development Kit
s	seconds
TC	terminal count
V	volts
WF	waveform
XMS	extended memory specification