DAQ

NI-DAQ® User Manual for PC Compatibles

Version 4.8

Data Acquisition Software for the PC

May 1995 Edition
Part Number 320498C-01

This document was created with FrameMaker 4.0.4

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

FTP Site: ftp.natinst.com

Bulletin Board Support

BBS United States: (512) 794-5422 or (800) 327-3077
BBS United Kingdom: (44) 635 551422
BBS France: (1) 48 65 15 59

FaxBack Support

(512) 418-1111 or (800) 329-7177

Telephone Support (U.S.)

Tel: (512) 795-8248
Fax: (512) 794-5678 or (800) 328-2203

International Offices

Australia (03) 879 9422, Austria (0662) 435986, 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 2637 5019,
Italy 02/48301892, Japan (03) 3788-1921, Korea 596-7456, Mexico 202 2544,
Netherlands 03480-33466, Norway 32-84 84 00, Singapore 2265886, Spain (1) 640 0085,
Sweden 08-730 49 70, Switzerland 056/20 51 51, Taiwan 62 377 1200, U.K. 1635 523545

National Instruments Corporate Headquarters

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

Important Information

Warranty

The media on which you receive National Instruments software are warranted not to fail to execute programming instructions, due to defects in materials and workmanship, for a period of 90 days from date of shipment, as evidenced by receipts or other documentation. National Instruments will, at its option, repair or replace software media that do not execute programming instructions if National Instruments receives notice of such defects during the warranty period. National Instruments does not warrant that the operation of the software shall be uninterrupted or error free.

A Return Material Authorization (RMA) number must be obtained from the factory and clearly marked on the outside of the package before any equipment will be accepted for warranty work. National Instruments will pay the shipping costs of returning to the owner parts which are covered by warranty.

National Instruments believes that the information in this manual is accurate. The document has been carefully reviewed for technical accuracy. In the event that technical or typographical errors exist, National Instruments reserves the right to make changes to subsequent editions of this document without prior notice to holders of this edition. The reader should consult National Instruments if errors are suspected. In no event shall National Instruments be liable for any damages arising out of or related to this document or the information contained in it.

EXCEPT AS SPECIFIED HEREIN, NATIONAL INSTRUMENTS MAKES NO WARRANTIES, EXPRESS OR IMPLIED, AND SPECIFICALLY DISCLAIMS ANY WARRANTY OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. CUSTOMER'S RIGHT TO RECOVER DAMAGES CAUSED BY FAULT OR NEGLIGENCE ON THE PART OF NATIONAL INSTRUMENTS SHALL BE LIMITED TO THE AMOUNT THERETOFORE PAID BY THE CUSTOMER. NATIONAL INSTRUMENTS WILL NOT BE LIABLE FOR DAMAGES RESULTING FROM LOSS OF DATA, PROFITS, USE OF PRODUCTS, OR INCIDENTAL OR CONSEQUENTIAL DAMAGES, EVEN IF ADVISED OF THE POSSIBILITY THEREOF. This limitation of the liability of National Instruments will apply regardless of the form of action, whether in contract or tort, including negligence. Any action against National Instruments must be brought within one year after the cause of action accrues. National Instruments shall not be liable for any delay in performance due to causes beyond its reasonable control. The warranty provided herein does not cover damages, defects, malfunctions, or service failures caused by owner's failure to follow the National Instruments installation, operation, or maintenance instructions; owner's modification of the product; owner's abuse, misuse, or negligent acts; and power failure or surges, fire, flood, accident, actions of third parties, or other events outside reasonable control.

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.

This document was created with FrameMaker 4.0.4

Table of Contents

About This Manual

Chapter 1
Introduction to NI-DAQ

This document was created with FrameMaker 4.0.4

Chapter 2
Hardware Overview

Chapter 3
Software Overview

Chapter 4
DMA and Programmed I/O Performance Limitations

Chapter 5
NI-DAQ Double Buffering

Customer Communication

Glossary

Index

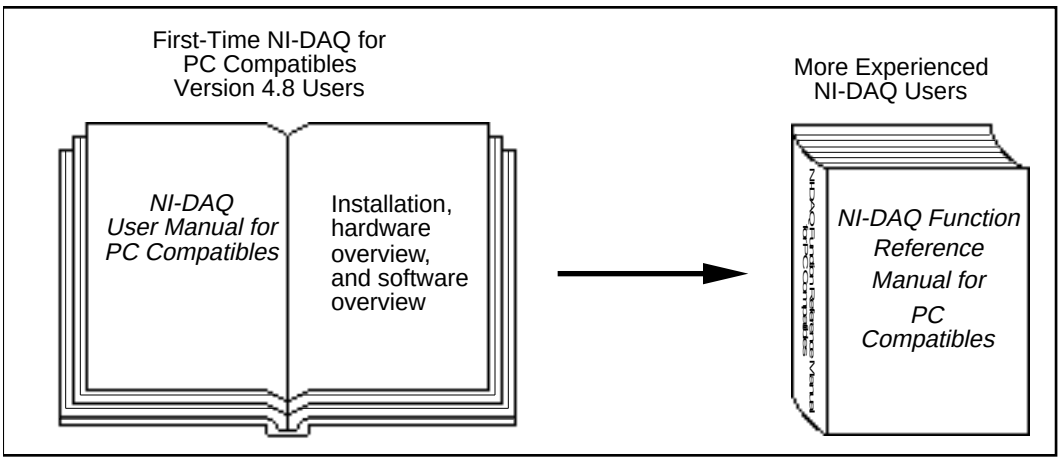
Figures

Tables

About This Manual

The *NI-DAQ User Manual for PC Compatibles* is for users of the NI-DAQ software for PC compatibles version 4.8. NI-DAQ software is a powerful application programming interface (API) between your data acquisition application and the National Instruments DAQ devices for ISA and EISA bus computers.

How to Use the NI-DAQ Manual Set



You should begin by reading the *NI-DAQ User Manual for PC Compatibles*. Chapter 1, *Introduction to NI-DAQ*, contains instructions on how to install and configure your National Instruments hardware and your NI-DAQ software. That chapter also 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

This document was created with FrameMaker 4.0.4

contains detailed descriptions of the NI-DAQ functions. Windows users can also use the Windows 3.1 help file `NIDAQFR.HLP`, which contains all of the function reference material.

Organization of This Manual

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

- Chapter 1, *Introduction to NI-DAQ*, describes how to install and configure NI-DAQ and the fundamentals of creating NI-DAQ applications in DOS, Windows, and Windows NT.
- Chapter 2, *Hardware Overview*, contains hardware information concerning your National Instruments DAQ device.
- Chapter 3, *Software Overview*, describes the classes of functions in NI-DAQ and briefly describes each function.
- Chapter 4, *DMA and Programmed I/O Performance Limitations*, discusses data acquisition performance reductions caused by interrupt latency in the Windows programming environment.
- Chapter 5, *NI-DAQ Double Buffering*, describes using double-buffered data acquisition with NI-DAQ.
- The *Customer Communication* appendix 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.
AI device	An analog input device has an *AI* in its name, such as the NEC-AI-16E-4.

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 parameters, menus, and error messages.
bold italic	Bold italic text denotes a note, caution, or warning.
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.
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.
`italic monospace`	Italic text in this font denotes that you must supply the appropriate words or values in the place of these items.
MIO device	The multifunction I/O devices have *MIO* in their names, such as the AT-MIO-16 and NEC-MIO-16E-4.
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 are to be literally input from the keyboard, sections of code, programming examples, and syntax examples. This font is also used for the proper names of disk drives, paths, directories, programs, subprograms, subroutines, device names, functions, 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.
PC	Refers to the IBM PC/XT, IBM PC AT, and compatible computers.
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 and SCXI-1001.
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*.

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-MIO-16	12-bit	AT	MIO	n/a
AT-MIO-16D	12-bit	AT	MIO	n/a
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	n/a
AT-MIO-16X	16-bit	AT	MIO	n/a
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	n/a

Device	Bit	Type	Functionality	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 User Manual for PC Compatibles* is one piece of the documentation set for your system. You could have any of several types of manuals, depending on the hardware and software in your system. Use the manuals you have as follows:

- *Getting Started with SCXI*—If you are using SCXI, this is the first manual you should read. It gives an overview of the SCXI system and contains the most commonly needed information for the modules, chassis, and software
- Your SCXI user manuals—These manuals contain 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.

☞ **Note:** ***Only NI-DAQ for PC compatibles versions 4.6.1 and earlier support LabWindows for DOS.***

- 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—These manuals contain maintenance information on the chassis, installation instructions, and information about making custom modules.

Related Documentation

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

- *Microsoft Visual C++ User Guide to Programming*
- *NIST Monograph 175*

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 the *Customer Communication* appendix at the end of this manual.

Introduction to NI-DAQ

Chapter 1

This chapter describes how to install and configure NI-DAQ and the fundamentals of creating NI-DAQ applications in DOS, Windows, and Windows NT.

About the NI-DAQ Software for PC Compatibles

Thank you for buying a National Instruments data acquisition (DAQ) device for the PC/XT/AT, EISA, or PCMCIA platforms, which includes NI-DAQ software for PC compatibles. NI-DAQ is a set of functions that control all of the National Instruments plug-in DAQ devices for analog I/O, digital I/O, timing I/O, SCXI signal conditioning, and RTSI multiboard synchronization.

NI-DAQ has both high-level *DAQ I/O* functions for maximum ease of use and low-level DAQ I/O functions for maximum flexibility and performance. Examples of high-level functions are streaming data to disk or acquiring a certain number of data points. Examples of low-level functions are writing directly to registers on the DAQ device or calibrating the analog inputs. NI-DAQ does not sacrifice the performance of National Instruments DAQ devices because it lets multiple devices operate at their peak performance.

NI-DAQ includes a *Buffer and Data Manager* that uses sophisticated techniques for handling and managing data acquisition buffers so that you can simultaneously acquire and process data. NI-DAQ can transfer data using DMA, interrupts, or software polling. NI-DAQ can use DMA to transfer data into memory above 16 MB even on ISA-bus computers.

With the NI-DAQ *Resource Manager*, you can simultaneously use several functions and several devices. The Resource Manager prevents multiple-board contention over DMA channels, interrupt levels, and RTSI channels.

NI-DAQ can send *event-driven messages* to DOS, Windows, or Windows NT applications whenever a user-specified event occurs.

This document was created with FrameMaker 4.0.4

Thus, polling is eliminated and you can develop event-driven DAQ applications. Some examples of NI-DAQ user events include when a specified number of analog samples has been acquired, when the analog level and slope of a signal match specified levels, when the signal is inside or outside a voltage band, when a specified digital I/O pattern is matched, and when a rising or falling edge occurred on a timing I/O line.

The DAQWare Utilities

DAQWare is a ready-to-run, Windows-based software system included with NI-DAQ for monitoring and controlling National Instruments PC/XT/AT, PCMCIA, and parallel port DAQ devices. DAQWare has an intuitive graphical user interface for configuring, monitoring, and controlling your DAQ devices. Analog, digital, and counter/timer I/O testing panels are built into the menu-driven system for fast, easy verification of the operation and capability of the board. You can use DAQWare for the following purposes:

- You can use the Analog Input, Analog Output, Digital Input/Output, and Counter/Timer test utilities contained in DAQWare to ensure that your DAQ hardware is working correctly.
- You can use the DAQWare instrument utilities to record voltages, generate waveforms, and measure temperature. These instruments are a Strip Chart Recorder and Data Logger, a Function Generator, and a Temperature Meter.
- You can use the configuration panels to choose from various settings of analog input and output. If you have an MIO device, you can also configure counters.
- The outputs of the Analog Output test, the Digital Output test, the Counter/Timer test, and the Function Generator instrument utilities remain unchanged until you change them. Therefore, you can generate output and then run an Analog Input and Digital Input test or the Strip Chart Recorder and Data Logger to verify or monitor the generated outputs.

Installing DAQWare

The NI-DAQ installation for Windows will automatically install DAQWare and create a DAQWare icon in your NI-DAQ program group. Refer the *NI-DAQ Installation for Windows* section later in this chapter.

☞ **Note:** ***You must install and configure your DAQ devices before running DAQWare. Refer to the*** *Device Installation and Configuration* ***section later in this chapter.***

DAQWare Hardware Support

DAQWare works with the National Instruments AI, MIO, DIO-24, Lab-PC+, PC-LPM-16, DAQCard-500, DAQCard-700, DAQCard-1200, and DAQPad-1200 devices. Refer to the `DAQWARE.TXT` file for a complete listing of supported devices.

NI-DAQ Distribution Media

The NI-DAQ software for PC compatibles are 3.5 in., 1.44 MB format floppy disks. Even if you do not have a 3.5 in. floppy disk drive on your computer, you may still be able to use the 3.5 in. diskettes under the following circumstances:

- If your computer has 5.25 in. floppy disk drives only but can share a remote network drive, you can use a computer that does have a 3.5 in. drive and copy the entire contents of the NI-DAQ software for PC compatibles disks onto the remote drive. You should create a directory on your remote drive with several subdirectories called `disk1`, `disk2`, `disk3`, and so on. Copy the entire contents of each NI-DAQ disk to the corresponding directory. Assuming your remote drive is `y:`, your directory structure would look like this:

  ```
  y:\nidaq\disk1
          \disk2
          \disk3
          ...
          \diskn
  ```

  If you are installing NI-DAQ for Windows, NI-DAQ for LabWindows/CVI, or NI-DAQ for LabVIEW, simply run `setupwin.exe` from the Windows Program Manager or File Manager by running `y:\nidaq\disk1\setupwin.exe`. If you are installing NI-DAQ for DOS, change your working directory to the directory `disk1`. Then run `setupdos.exe` with the `-d` flag (i.e. `setupdos -d`) and the DOS installer will be able to find the remaining disks in the `disk2`, `disk3`, and so on directories. Be sure to change the source drive option in `DAQCONF` so that it corresponds to your remote drive.

- If you are installing NI-DAQ for DOS but cannot use a remote network drive, you can install NI-DAQ on a computer with 3.5 in. floppy drives. All of the NI-DAQ files will be installed in the `NIDAQDOS` directory and its subdirectories. Use a file compression utility (such as pkzip) to compress all of the files in the `NIDAQDOS` directory and its subdirectories. Then transfer the compressed file via ftp or a mail program that allows you to enclose files, or, if the computer also has 5.25 in. drives, via 5.25 in. floppy disks.

  Once you have transferred the compressed file onto your destination computer, uncompress the file. This method is only recommended for NI-DAQ for DOS because NI-DAQ for DOS is the only case where all the installed files are contained in one directory structure.

  You may also be able to copy selected files from the 3.5 in. diskettes onto 5.25 in. diskettes to produce diskettes that can install either NI-DAQ for DOS or NI-DAQ for Windows. To create 5.25 in. DOS-installable diskettes, copy files from the 3.5 in. diskettes to 5.25 in. diskettes as follows:

  – `disk1`: Copy the files `setupdos.exe`, `nidaqdos.dat`, and all `.zip` files
  – All other disks: Copy all `.zip` files.

  To create 5.25 in. Windows installable diskettes, copy all files except those listed for the DOS installation onto the corresponding 5.25 in. diskettes.

If you cannot use any of these methods, please call National Instruments and request part number 463276-40 if you need 5.25 in. diskettes for NI-DAQ for PC compatibles.

How to Set Up Your DAQ System

Figure 1-1 shows the steps to install your hardware and software, configure your hardware, and begin using NI-DAQ in your application programs.

If you will be accessing the NI-DAQ device drivers through LabVIEW, you should read the *NI-DAQ Installation for LabVIEW* section, then use your *LabVIEW for Windows Data Acquisition VI Reference Manual* to help you get started using the data acquisition VIs in LabVIEW.

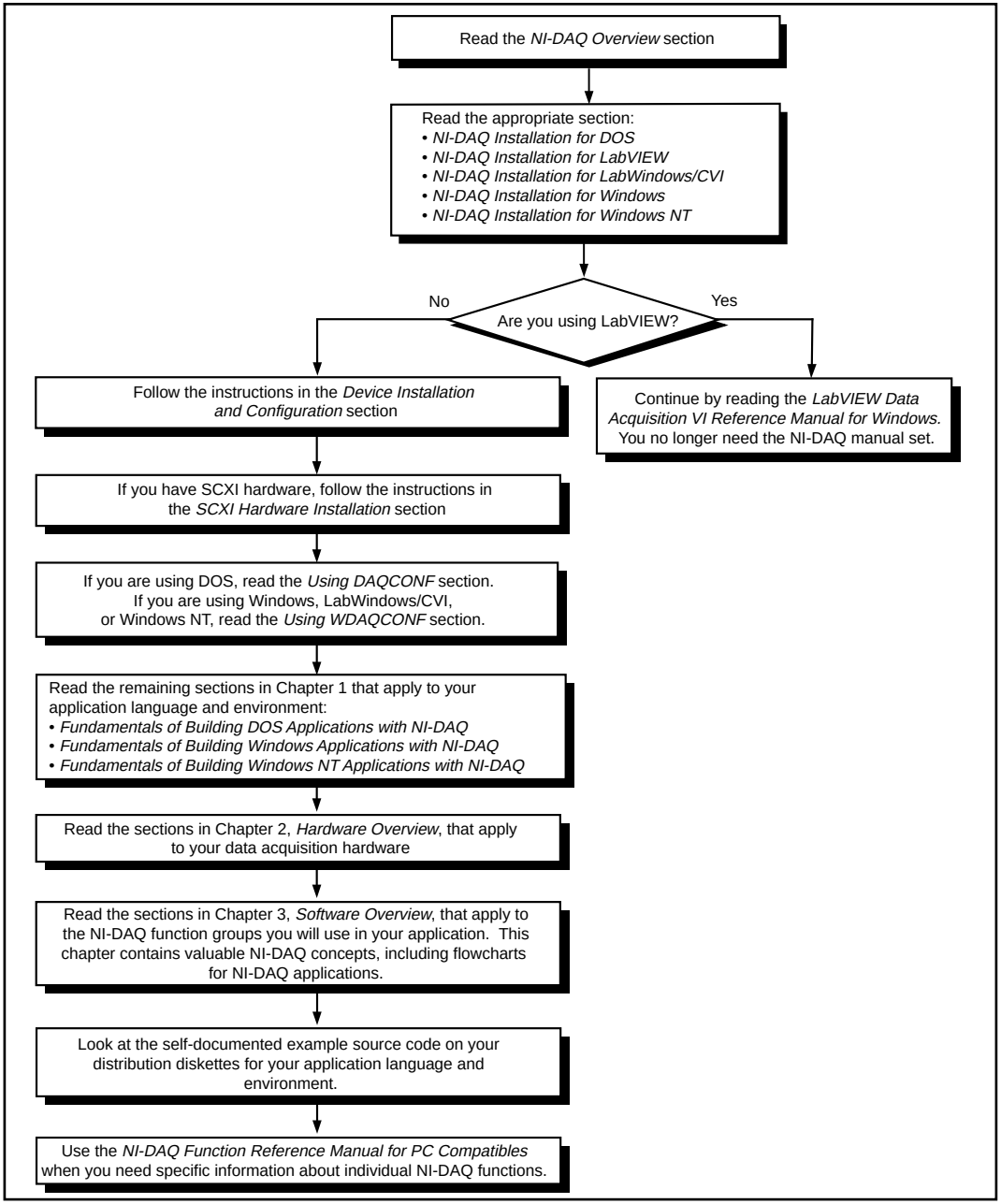


Figure 1-1. How to Set up Your DAQ System

NI-DAQ Overview

NI-DAQ is a library of routines that work with National Instruments DAQ devices. NI-DAQ contains services for overcoming difficulties ranging from simple device initialization to advanced high-speed data logging. The number of services you need for your applications depends on the types of DAQ devices you have and the complexity of your applications.

NI-DAQ Hardware Support

National Instruments periodically upgrades NI-DAQ to add support for new DAQ hardware. To ensure that this version of NI-DAQ supports the hardware you are going to use, consult Table 1-1.

Table 1-1. NI-DAQ Hardware Support

AT Series[1]	PC Series[1]	EISA	NEC[1]	External Device[1]	PCMCIA[1]	SCXI[1]
AT-A2150	Lab-PC+	EISA-A2000	NEC-AI-16E-4	AMUX-64T	DAQCard-500	SCXI-1000
AT-AO-6/10	PC-AO-2DC		NEC-AI-16XE-50	DAQPad-1200[2]	DAQCard-700	SCXI-1001
AT-DIO-32F	PC-DIO-24		NEC-MIO-16E-4	DAQPad-MIO-16XE-50[2]	DAQCard-1200	SCXI-1100
AT-DSP2200	PC-DIO-96		NEC-MIO-16XE-50	SC-2040	DAQCard-AO-2DC	SCXI-1102
AT-MIO-16	PC-LPM-16			SC-2042-RTD	DAQCard-DIO-24	SCXI-1120
AT-MIO-16D	PC-OPDIO-16			SC-2043-SG		SCXI-1121
AT-MIO-16DE-10	PC-TIO-10					SCXI-1122
AT-MIO-16E-1						SCXI-1124
AT-MIO-16E-2						SCXI-1140
AT-MIO-16E-10						SCXI-1141
AT-MIO-16F-5						SCXI-1160
AT-MIO-16X						SCXI-1161
AT-MIO-16XE-50						SCXI-1162
AT-MIO-64E-3						SCXI-1162HV
AT-MIO-64F-5						SCXI-1163
						SCXI-1163R
						SCXI-1200[2]

[1]NI-DAQ for Windows NT does not work with the SCXI-1102, SCXI-1122, SCXI-1124, SCXI-1141, SCXI-1162HV, SCXI-1163R, SCXI-1200, DAQPad-1200, PC-AO-2DC, PCMCIA cards, E Series devices, SC-2040, SC-2042-RTD, SC-2043-SG, and PC-OPDIO-16.

[2]The SCXI-1200, DAQPad-MIO-16XE-50, and DAQPad-1200 do not work with NEC PC-9800 computers.

Throughout this manual, many of the devices are grouped into categories that are similar in functionality. The categories are often used in the text to avoid long lists of specific devices. The *Conventions Used in This Manual* section of *About This Manual* lists the devices in each functional type. Any device not included in a category will always be referred to in the manual by its name.

NI-DAQ Language Support

NI-DAQ supports the following languages under DOS:

- Microsoft Visual C++ 1.0 and 1.5
- Turbo C++ 3.0
- Borland C++ 3.0, 3.1, 4.0, and 4.5
- Borland Turbo Pascal 7.0 (real mode)
- Visual Basic for DOS

NI-DAQ supports the following languages under Windows:

- Microsoft Visual C++ 1.0 and 1.5
- Visual Basic 1.0, 2.0, and 3.0
- Turbo Pascal for Windows 1.0 and 1.5
- Turbo Pascal 7.0
- Borland C++ 3.0, 3.1, 4.0, and 4.5

NI-DAQ supports the following language under Windows NT:

- Microsoft Visual C++ 1.0 and 2.0 (32-bit editions only)

Most of the files on the release diskettes are compressed. Always run the NI-DAQ installation utilities to extract the files you want. For a brief description of the directories produced by the install programs and the names and purposes of the uncompressed files, consult the `README.DAQ` file.

NI-DAQ Installation for DOS

The NI-DAQ distribution diskettes contain the installation utility `SETUPDOS.EXE`. Running this installation utility copies the appropriate files to your computer. For example, if your installation diskette is in drive `A`, type the following:

```
a:\setupdos
```

After installing NI-DAQ, you must install your DAQ devices. Continue by reading the *Device Installation and Configuration* section later in this chapter.

NI-DAQ Installation for LabVIEW

The LabVIEW installation program may have installed the NI-DAQ software for you. However, the NI-DAQ software included with your DAQ hardware may be a more recent revision than the NI-DAQ software that LabVIEW installed.

After you have installed LabVIEW, run the NI-DAQ Windows installer `SETUPWIN.EXE`, which will check the NI-DAQ version that LabVIEW installed against this NI-DAQ version to ensure that the newest version is installed.

To upgrade NI-DAQ for LabVIEW, run the `SETUPWIN` program on Disk 1. One way to do this is to select the **File** menu from the Program Manager Window, then select **Run...** and type in `a:\setupwin`, assuming `a:` is the floppy disk drive containing Disk 1. When prompted, select the **Upgrade NI-DAQ for LabVIEW** option.

The NI-DAQ installer examines your computer system to determine the system-dependent files that you need. For a description of the modifications the NI-DAQ installer has made to your `SYSTEM.INI` file, see the *NI-DAQ Modifications to the SYSTEM.INI File* section later in this chapter.

Depending on your LabVIEW version, it may be necessary for NI-DAQ to update some of the LabVIEW data acquisition VIs. If so, carefully follow the instructions given in the NI-DAQ installer and the `README.DAQ` file.

Continue by reading your *LabVIEW for Windows Data Acquisition VI Reference Manual*. You no longer need this manual set.

NI-DAQ Installation for LabWindows/CVI

To install NI-DAQ for LabWindows/CVI, run the `SETUPWIN` program on Disk 1. One way to do this is to select the **File** menu from the Program Manager Window, then select **Run...** and type in `a:\setupwin`, assuming `a:` is the floppy disk drive containing

Disk 1. When prompted, select the **Install NI-DAQ for LabWindows/CVI** option.

The NI-DAQ example programs for LabWindows/CVI are installed in the `CVI\SAMPLES\DAQ` directory.

For LabWindows/CVI, the defined constants that several NI-DAQ functions use are in the include file `DATAACQ.H`.

The NI-DAQ installer examines your computer system to determine the system-dependent files that you need. For a description of the modifications the NI-DAQ installer has made to your `SYSTEM.INI` file, see the *NI-DAQ Modifications to the SYSTEM.INI File* section later in this chapter.

After you complete the NI-DAQ for LabWindows/CVI installation, you must install and configure your DAQ devices. Continue by reading the *Device Installation and Configuration* section later in this chapter.

☞ **Note:** ***If you are using an AT-DSP2200 with LabWindows/CVI, you should install your NI-DSP for DOS disks to obtain the files `DSP2200.OUT` and `DSP2200S.OUT`. These files are necessary to configure the AT-DSP2200 in the `WDAQCONF` configuration utility. None of the other files installed by NI-DSP for DOS will be used and can be deleted after installation.***

NI-DAQ Installation for Windows

To install NI-DAQ for Windows, run the `SETUPWIN` program on Disk 1. One way to do this is to select the **File** menu from the Program Manager Window, then select **Run...** and type in `a:\setupwin`, assuming `a:` is the floppy disk drive containing Disk 1. When prompted, select the **Install/Upgrade NI-DAQ for Windows** option.

`Setupwin` will install examples programs and support files for a variety of languages and compilers. Choose all of the languages/compilers you plan to use.

The NI-DAQ installer examines your computer system to determine the system-dependent files that you need. For a description of the modifications the NI-DAQ installer has made to your `SYSTEM.INI` file, see the *NI-DAQ Modifications to the SYSTEM.INI File* section following this section.

After you complete the NI-DAQ for Windows installation, you must install and configure your DAQ devices. Continue by reading the *Device Installation and Configuration* section later in this chapter.

NI-DAQ Modifications to the SYSTEM.INI File

Because of large interrupt latencies in Windows 386 enhanced mode, National Instruments has developed a Windows virtual driver that handles NI-DAQ interrupt services. The installer places this file, `NIVISRD.386`, in the `WINDOWS\SYSTEM` directory. The installer also modifies the `SYSTEM.INI` file to load and use the `NIVISRD.386` driver by adding the following statement to the `[386Enh]` section of the `SYSTEM.INI` file:

```
device=NIVISRD.386
```

☞ **Note:** `NIVISRD.386` ***is an optional performance-enhancing supplement—you do not have to install it to run NI-DAQ for Windows applications. To deactivate*** `NIVISRD.386`***, comment out the statement inserted by the installer in your*** `SYSTEM.INI` ***file by placing a pound sign (#) before the line.***

If you have an EISA-A2000 application and do not have `NIVISRD.386` installed, you will encounter the following constraints:

- NI-DAQ cannot report logical scans done in pretriggered data acquisition.
- Your data acquisition buffers must be contiguous in physical memory.

After you install NI-DAQ, read the *Device Installation and Configuration* section later in this chapter for instructions on installing your plug-in devices.

NI-DAQ Installation for Windows NT

☞ **Note:** ***NI-DAQ for Windows NT does not work with the SCXI-1102, SCXI-1122, SCXI-1124, SCXI-1141, SCXI-1162HV, SCXI-1163R, SCXI-1200, PCMCIA cards, E Series devices, SC-2040, SC-2042, SC-2043, and AO-2DC devices.***

Installation Utility

NI-DAQ for Windows NT has an installation utility to copy all the files you need to develop NI-DAQ Windows NT applications. Since all the NI-DAQ files are compressed on the installation diskettes, you must run the installation utility to correctly install NI-DAQ for Windows NT.

To install NI-DAQ for Windows NT, start Windows NT and log into Windows NT as an administrator or as a user with administrator privilege. Select `Run` under the `File` menu in the Program Manager and type the following command:

```
z:\setupwin
```

where `z` is the drive letter of the diskette drive.

Follow the installation utility prompts as you go through the installation process. When you are done, make sure you reboot Windows NT by selecting the `Shutdown` option under the `File` menu in the Program Manager.

When the NI-DAQ for Windows NT installation is done, the installation program will attempt to modify your system registry so that the NI-DAQ for Windows NT device driver can be loaded into the system when necessary. In the registry, the installation program creates two new keys. The first key is as follows:

```
\\HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Service\NIDAQNT
```

Under this entry, there will be three new values as follows:

```
ErrorControl:REG_DWORD:0x1
Start:REG_DWORD:0x3
Type:REG_DWORD:0x1
```

The second key is as follows:

```
\\HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Service\NIDAQKNL
```

Under this entry, there will be three new values as follows:

```
ErrorControl:REG_DWORD:0x1
Start:REG_DWORD:0x1
Type:REG_DWORD:0x1
```

If the installation program fails to update the registry, you must create them with REGEDT32.EXE in your Windows NT directory. To create the keys manually, go to the Program Manager, select the **Run** option under the **File** menu, and type:

```
regedt32
```

The program you are running is called the Registry Editor. In the main window, you will see four child windows. Select the child window named HKEY_LOCAL_MACHINE on the local machine. In the HKEY_LOCAL_MACHINE child window, select **SYSTEM** by double-clicking on it. Select **CurrentControlSet** by double-clicking on it. Select **Services** by double-clicking on it. After you have made all the selections, select the **Add Key** option under the **Edit** menu.

When the **Add Key** dialog box pops up, enter NIDAQNT for the **Key Name**. Then, click on the OK button.

You should now see the new key, NIDAQNT, under the **Services** registry key. All the keys are alphabetized; you may have to scroll down the window to see the **Services** key. When you find it, select **NIDAQNT** by clicking on it once.

Now, select **Add Value** under the **Edit** menu. When the **Add Value** dialog box pops up, enter ErrorControl for the **Value Name**. Set **Data Type** to REG_DWORD. When you click the OK button, a new dialog box will pop up and prompt you for a value. Enter 1 and click the OK button. You have now entered a new value under NIDAQNT.

Repeat the **Add Value** procedure for the other two new values, **Start** and **Type**. You should set the **Start** value to 3 and the **Type** value to 1. When you are done, make sure all three values under registry key **NIDAQNT** are as follows:

```
ErrorControl:REG_DWORD:0x1

Start:REG_DWORD:0x3

Type:REG_DWORD:0x1
```

If you make a mistake, you can remove the value or the key by selecting it and pressing the **Delete** key. Be careful not to delete any other key in the registry.

After you add NIDAQNT into your registry, repeat the same procedure for NIDAQKNL. Notice that the start value is 0x1 for NIDAQKNL.

Notice that NI-DAQ for Windows NT only works with National Instruments ISA and EISA DAQ devices on *x*86-based computers.

After you install NI-DAQ, read the *Device Installation and Configuration* section for instructions on installing your plug-in devices.

Device Installation and Configuration

Before you begin your NI-DAQ application development, you must configure your National Instruments DAQ devices, which can be plug-in devices, PCMCIA cards, or external devices you connect to your computer's parallel port. NI-DAQ needs the device configuration information to program your hardware properly.

Because all system architectures are different, each requires a different device configuration procedure. This ensures that your DAQ devices will work properly and coexist with other peripherals in your system such as serial ports and parallel ports.

Before installing your DAQ devices, consult your hardware user manual to see if you need to change any hardware-selectable options. Some DAQ devices have jumpers to select analog input polarity, input mode, analog output reference, and so on. Be sure to make a note of which device options you change, so that you can notify NI-DAQ either by entering the information in one of the NI-DAQ configuration utilities or using NI-DAQ function calls in your application.

Some DAQ devices also have jumpers to select IRQ levels and/or DMA channels. If you have multiple DAQ devices in your system, you should try to select different IRQ levels and DMA channels for each device. You should also select a unique base address for each device if your device has address DIP switches. Make a note of your device IRQ, DMA, and address settings.

The installation and configuration process you need to follow depends on your computer system and the type of DAQ product you have. The following sections provide information on the configuration procedure for PCMCIA cards, DAQPads, switchless devices, and plug-in DAQ devices for ISA (PC AT/XT) and EISA computers.

Configuring PCMCIA DAQ Cards

After inserting your PCMCIA DAQ card in your computer, you must run the NI-DAQ configuration utility; read the *Using DAQCONF* section if you are using DOS, and read the *Using WDAQCONF* section if you are using Windows or LabWindows/CVI. If you have SCXI hardware, you should read the *SCXI Hardware Installation* section later in this chapter next.

Configuring Parallel Port DAQ Devices (DAQPads)

If you are installing a parallel port DAQ device (known as a DAQPad), connect one end of the parallel port cable to the DAQPad. Connect the other end to the parallel port on your PC.

Then you must run the NI-DAQ configuration utility; read the *Using DAQCONF* section if you are using DOS, or read the *Using WDAQCONF* section if you are using Windows or LabWindows/CVI.

Configuring Plug and Play (Switchless) Devices

☞ **Note:** ***We recommend that you configure all non-Plug and Play devices in your system before adding any Plug and Play devices. This enables the configuration utility to perform better resource checking on all your DAQ devices.***

The National Instruments E Series switchless devices support switchless and jumperless (Plug and Play) configuration in DOS and Windows. All resources including base address, DMA channels, and IRQ levels on these devices are fully software configurable. No jumpers or DIP switches are needed to configure any of these resources.

The NI-DAQ installer will install a stand-alone executable called `NI-PNP.EXE` in the boot directory of your root drive. This program detects and configures any Plug and Play devices you have in your computer. The program will run every time you boot from your `autoexec.bat` file. After configuring your Plug and Play hardware in the system, the program will generate an `NI-PNP.INI` file in the same directory. This file contains information about the National Instruments devices in your system, including Plug and Play devices.

The DAQ configuration utility (`WDAQCONF` or `DAQCONF`) will read the `NI-PNP.INI` for information and will automatically configure any Plug and Play devices you have in your computer. The utility will

also deconfigure any previously configured Plug and Play device that you have removed from your computer. Running the configuration utility after installing a new Plug and Play device is important because you will be able to obtain a mapping for the newly installed device into an NI-DAQ device number.

When the configuration utility finds a new Plug and Play device in your computer, it assigns the first available device number to the new device. The utility also assigns default resources such as I/O address, DMA channels, or IRQ levels to the new device. When you remove the device from your computer, the utility deallocates these resources and the device number will contain an "empty device."

☞ **Note:** ***You must run the DAQ configuration utility after you install or remove any National Instruments Plug and Play devices.***

If you have Plug and Play software in your system, the behavior of the DAQ configuration utility may change significantly. *If the Plug and Play software in your system has its own separate configuration utility, you must use the system configuration utility to configure all National Instruments devices in your system.* Subsequently, you must run the DAQ configuration utility in order to assign NI-DAQ device numbers to any new devices. If you do not run the DAQ configuration utility in this case, you will be unable to configure any system resources for the device such as base address, DMA channels, or IRQ levels. The configuration utility that comes with your Plug and Play software is responsible for assigning system resources to your National Instruments device.

Examples of Plug and Play software are a Plug and Play BIOS or the Intel Plug and Play Kit, which includes the Intel Configuration Manager and has its own configuration utility (ICU).

`WDAQCONF` performs a full set of tests before saving the device configuration to ensure the device will operate correctly. If the device fails any of the tests, `WDAQCONF` reports the errors and does not save the configuration.

`DAQCONF` does not perform any such tests. The only way to find out if the configuration is 100% successful in DOS is to run a few NI-DAQ calls on the device. We recommend you try the `AI_Read` and `DAQ_Op` calls first. Remember to use the `Timeout_Config` call before calling `DAQ_Op` so that your computer will not lock up if the call fails to return any data.

Configuring DAQ Devices on ISA (PC AT/XT) Computers

Industry Standard Architecture (ISA) is the oldest computer system architecture among the three system architectures NI-DAQ supports. The ISA specification does not define any standard device setup procedure. It relies on the computer user to make sure that all the plug-in devices are free of resource conflicts, no plug-in device I/O base address ranges are overlapping each other, and no plug-in devices are using the same interrupt levels.

After you check your device jumpers and DIP switches, turn off your computer and plug in your devices. Then continue with the *SCXI Hardware Installation* section (if applicable), *Using DAQCONF* (DOS users), or *Using WDAQCONF* (Windows users).

Configuring DAQ Devices on EISA Computers

☞ **Note:** ***You do not use the System Configuration Utility supplied by your EISA computer vendor to configure any device except those listed in Table 1-3. Use only the appropriate National Instruments configuration utility,*** `DAQCONF.EXE` ***for DOS or*** `WDAQCONF.EXE` ***for Windows.***

Extended Industry Standard Architecture (EISA) is a superset of the ISA architecture. All National Instruments AT/XT Series DAQ devices will operate properly on an EISA bus computer. However, unlike the ISA specification, EISA provides a standard way to keep track of plug-in device configurations on the system CMOS memory. Furthermore, all EISA computers are packaged with EISA configuration utilities to configure the computers. Although EISA specifies a standard way to store and to retrieve system configuration, it does not specify a standard configuration utility. Therefore, this manual includes only a general procedure on how to install National Instruments plug-in devices.

Before using the EISA configuration utility, you must know the resource settings of your devices. You can obtain the information from the DIP switches and jumpers on your devices. If your DAQ devices do not have I/O base address DIP switches or jumpers for interrupt levels and DMA channels, obtain as much information as you can; other resource settings are software programmable. If you have trouble reading DIP switches or jumper values, consult your DAQ device user manuals for assistance.

After you have checked your device jumpers and DIP switches, follow these general instructions to install National Instruments plug-in devices:

1. Launch the EISA configuration utility. Some EISA computer manufacturers put the utility on a floppy disk or on the computer hard disk. If the utility is on your hard disk, it is probably under the directory `\ECU` or `\CONFIG`. The utility file is usually named `CF.EXE` or `CFG.EXE`. When you find the directory where the utility is, copy all National Instruments EISA configuration files into the same directory. You can find National Instruments EISA configuration files in the directory `EISACFG` under the directory you select when you install NI-DAQ.

   Some EISA computer manufacturers put the EISA configuration utility on your hard disk, but your operating system cannot directly read the utility. In this case, you can probably launch the configuration utility by pressing a certain key(s) when the computer is powered up. On some computers, you can press the F10 key right after the machine makes the power-on beep.

   If you do not know how to launch your EISA configuration utility, consult your computer user manuals or call your computer manufacturer's technical support.

2. When you have the EISA configuration utility running, follow all the prompts and instructions on how to configure a plug-in device. Look for the option on adding a device.

   If your EISA computer manufacturer put the EISA configuration utility on your hard disk where your operating system cannot directly read the utility, you must instruct the utility to get the National Instruments EISA configuration files from the directory `EISACFG` in the directory in which you installed NI-DAQ.

3. After you successfully add a device, look for the change resource option. You can use this utility to check whether the default resource setting matches your real device setting. Notice that the software default setting is not necessarily the same as the factory hardware default setting. Change either the software setting in the configuration utility or the hardware setting on your device so that they match.

4. When you have your DAQ device correctly configured, save the configuration and reboot your computer.

5. Turn off your computer and install the DAQ device. Consult your computer user manuals on how to install a plug-in device into your computer.

After you install your plug-in devices, if you have SCXI hardware, you should read the *SCXI Hardware Installation* section later in this chapter. After you install your SCXI hardware, if any, you should read the *Using DAQCONF* section if you are using DOS. You should read the *Using WDAQCONF* section if you are using Windows or LabWindows/CVI.

Table 1-2 shows which configuration file you should use for your DAQ device.

Table 1-2. EISA Configuration Files

DAQ Device	EISA Configuration File
AT-A2150	!NIC1000.CFG
AT-AO-6/10	!NIC1200.CFG
AT-DIO-32F	!NIC0301.CFG
AT-DSP2200	!NIC1100.CFG
AT-MIO-16	!NIC0202.CFG
AT-MIO-16D	!NIC1500.CFG
AT-MIO-16F-5	!NIC0602.CFG
AT-MIO-16X	!NIC1300.CFG
AT-MIO-64F-5	!NIC1400.CFG
EISA-A2000	!NIC0101.CFG
Lab-PC+	!NIC0501.CFG
PC-DIO-24	!NIC0400.CFG
PC-DIO-96	!NIC0700.CFG

Table 1-2. EISA Configuration Files (Continued)

DAQ Device	EISA Configuration File
PC-LPM-16	!NIC0800.CFG
PC-TIO-10	!NIC0900.CFG

☞ **Note:** ***There are no EISA configuration files for PCMCIA DAQ devices, parallel port devices, and switchless devices. You can configure these devices directly from the appropriate configuration utility.***

SCXI Hardware Installation

The *Getting Started with SCXI* manual that came with your SCXI hardware contains step-by-step detailed instructions for assembling your SCXI system, including module jumper settings, cable assemblies, and terminal blocks. The basic steps are as follows:

1. Check the jumpers on your modules. You should almost always leave the jumpers in the default positions. The *Getting Started with SCXI* manual has a section for each module type that lists the cases in which you may want to change the jumper settings. The SCXI-1120, SCXI-1121, and SCXI-1140 modules have jumper-selectable gains for each channel.
2. Make sure that the chassis power is turned off. Plug your modules in through the front of the chassis. You can put the modules in any slot; for simplicity start with slot 1 on the left side of the chassis and move right with additional modules. Be sure to screw the modules tightly into the chassis frame.
3. If you are using an SCXI-1180 feedthrough panel, you must install the SCXI-1180 in the slot immediately to the right of the module that you will cable to the DAQ device. Otherwise, the cable connectors may not fit together conveniently.
4. Plug the appropriate terminal blocks into the front of each module and screw them tightly into the chassis frame.
5. If you have more than one chassis, select a unique jumpered address for each additional chassis by using the jumpers directly behind the front panel of the chassis.
6. If you are using a plug-in DAQ device with your SCXI system, connect the mounting bracket of the SCXI-134*x* cable assembly to the back of one of the modules and screw it into the chassis frame. Connect the other end of the cable to the DAQ device in your

computer. In Multiplexed mode, you only need to cable one module to the DAQ device, and in most cases it does not matter which module. There are two special cases:

- If you are using SCXI-1140 modules along with other types of modules, you need to cable one of the SCXI-1140 modules to the DAQ device.
- If you are using analog input modules and digital modules, you need to cable one of the analog input modules to the DAQ device.

Refer to the *Getting Started with SCXI* manual for more information on related topics, such as multichassis cabling.

7. If you are using the SCXI-1200 module, connect one end of the parallel port cable to the SCXI-1200 module. Connect the other end to the parallel port on your PC.
8. Turn on your chassis power.

Using DAQCONF

DAQCONF is a DOS-based application that you can use to view and configure your DAQ devices and SCXI hardware for NI-DAQ to use. You need to run DAQCONF if you are using NI-DAQ in DOS. If you are using NI-DAQ in Windows, LabWindows/CVI, or Windows NT, you should skip to the *Using WDAQCONF* section later in this chapter.

Locate DAQCONF in the same directory in which you installed NI-DAQ using the installation program. Run the configuration utility by typing DAQCONF at the DOS prompt.

NI-DAQ Configuration File

On ISA (PC AT/XT) computers, the NI-DAQ configuration file holds all configuration information for your DAQ hardware. On EISA computers, the system configuration utility holds basic configuration information for each plug-in device (such as I/O base addresses, IRQ levels, and DMA channels), and the NI-DAQ configuration file holds all other configuration information (such as SCXI configuration, DSP kernel file path, and PCMCIA configuration).

The NI-DAQ configuration file in DOS is named ATBRDS.CFG. The first time you run DAQCONF, it will create ATBRDS.CFG in your root directory. If you wish to create the configuration file in a different

directory, provide a path name when you run DAQCONF as in the following example:

```
DAQCONF \PROJ_X
```

With this option, you can create multiple configuration files for different NI-DAQ applications or projects; simply use the appropriate path name when you want to create a new configuration file or view an existing one. Be sure to enter only the path name; the file will automatically be created as ATBRDS.CFG in the specified directory.

When you run an NI-DAQ DOS application, NI-DAQ will look for the configuration file in the current directory first. If NI-DAQ cannot find ATBRDS.CFG, it will look in the root directory of the current drive. NI-DAQ will also read the device configuration that was stored in the EISA system configuration utility.

Device Configuration in DAQCONF

DAQCONF opens with the device configuration panel. If you are on an ISA (PC AT/XT) computer, perform the following steps to enter device number, I/O base address, IRQ levels, and DMA channels for your devices. If you are on an EISA computer, DAQCONF will read that information from the system configuration utility and display it for you.

If you have a DSP board or PCMCIA card, you must perform the following steps regardless of what type of computer you are using.

1. Select a **Device Number** for your device. Use the F5 and F6 keys to scroll through the choices. If the device number selected has a device assigned to it, you will see the current settings for that device. To add a device, select a number without any device assigned to it. On EISA computers, the device number for each device is determined by the slot number that you assigned in the system configuration utility.

   You will use the device number in your NI-DAQ function calls to identify which device you want to use.

2. Use the down arrow key to highlight the **Device** selection. Use the F5 and F6 keys to find the correct device type.

☞ **Note:** ***Switchless devices are automatically assigned a device number when*** DAQCONF ***launches. Therefore, you will not see switchless devices in the*** **Device Selection** ***list. Refer to the*** *Configuring Switchless Devices* ***section***

earlier in this chapter for more information on how* DAQCONF *configures switchless devices.

3. If you are on an ISA (PC AT/XT) computer, you need to select the correct I/O base address, IRQ levels, and DMA channels. Use the up/down arrow keys to highlight the fields, and then use the F5 and F6 keys to select the correct values that match your hardware jumpers. The initial values displayed correspond to the factory default settings; you only need to change these settings if you have changed the jumper settings on your device. If your device does not have jumpers, select the settings you wish to use.

   For devices with I/O base address DIP switches, DAQCONF will display a set of switches for that field. Use the right/left arrow keys to highlight each switch, then use the F5 and F6 keys to turn each switch on or off to match the settings on your device.

   For PCMCIA cards, you can select **Auto assign** for the I/O base address, IRQ levels, and DMA channels. If you select **Auto assign**, NI-DAQ will select these settings automatically at application run time based on the resource allocation in the system at that time.

4. If you are configuring an AT-DSP2200 board, you need to enter the path for your kernel file in the **Kernel Path** field. Consult your NI-DSP documentation for the name and location of your kernel file.

5. If you are configuring a PCMCIA card such as the DAQCard-700, you must enter the PCMCIA socket.

6. You must save the configuration for this device before advancing to the next device number. Press F10 to save. DAQCONF will test the configuration parameters before saving. If the test fails, DAQCONF will not save the settings. You can disable the automatic test feature by using the `-t` option on the command line when you invoke DAQCONF as in the following example:

   ```
   DAQCONF -t
   ```

7. After you configure all of your devices, if you have SCXI hardware, press F2 to bring up the SCXI configuration panel and follow the instructions in the next section, *SCXI Configuration in DAQCONF*.

   If you do not have SCXI hardware, press Esc to quit DAQCONF. Continue by reading the sections appropriate to your specific programming language and environment later in this chapter for fundamentals on building NI-DAQ applications.

SCXI Configuration in DAQCONF

1. Leave the **Chassis ID** set to one. You will use this number to access the SCXI chassis in the NI-DAQ function calls. If you have multiple chassis, advance the **Chassis ID** using the F6 key to configure the next chassis after you have configured the first chassis.
2. Use the down arrow key to highlight **Chassis Type**. Select the correct chassis type using the F5 and F6 keys.
3. If you have only one chassis, leave the **Chassis Address** field and the address jumpers on your SCXI chassis set to zero. If you have additional chassis, you need to select a unique hardware-jumpered address for each chassis and enter it in the **Chassis Address** field.
4. Enter the configuration for each slot in the chassis. The slot configuration fields are indented on the panel. The SCXI slots are numbered left to right when you are looking at the front of the chassis, beginning with slot 1. For each slot, enter the following information (use the up/down arrow keys to move from one field to another):
   a. **Module Type**. Select the correct module type that you have installed in this slot using the F5 and F6 keys. If you did not install a module in this slot, leave this field at **None** and advance the **Module Slot** number to the next slot.
   b. If the module in this slot is *directly* cabled to a DAQ device in your computer, set the **Cabled Device** field to the **Device Number** for that device. Leave the **Cabled Device** field at **None** if the module in the current slot is not directly cabled to a DAQ device in your PC.
   c. **Operating Mode**. Multiplexed mode is the default operating mode—it is recommended for almost all SCXI applications. The operating modes available for each SCXI module type are discussed in *The SCXI Hardware* section of Chapter 2, *Hardware Overview*.

      If you are operating your modules in Multiplexed mode, you only need to cable one module in each chassis to your plug-in DAQ device in the PC. If you are using an SCXI-1200 module in your chassis, you do not need to cable any modules in your chassis to a DAQ device in the PC; the SCXI-1200 can control the chassis.

d. If the module in this slot is an SCXI-1200, there are three additional fields:

Logical Device Number. You must assign the SCXI-1200 a logical device number so that you can use the SCXI-1200 with the NI-DAQ functions as if it were a plug-in DAQ device in the PC. The SCXI-1200 has the same functionality as the Lab-PC+ plug-in board. After you finish your SCXI configuration, if you press F2 to go back to the device configuration panel, the SCXI-1200 will be configured for this **Logical Device Number** in the device configuration panel.

Connected to… is the parallel port address to which you have cabled the SCXI-1200.

IRQ Level is the interrupt level you want the parallel port interrupt service routines to use. During a data acquisition or waveform generation on the SCXI-1200, NI-DAQ services parallel port interrupts to transfer data between the SCXI-1200 and your PC memory. This IRQ level must correspond to the IRQ level used by the parallel port to which the SCXI-1200 is connected.

5. After you have entered the configuration for each slot, look at the **Communication Mode** field. This field should set itself automatically to **Serial communication using DIO port** if you have cabled one of your modules to a DAQ device in the PC. This means that NI-DAQ will use a DIO port of the DAQ device to communicate serially with the SCXI hardware. If you have an SCXI-1200 in your chassis, this field should be **Parallel communication using LPT**, meaning that NI-DAQ will communicate with the SCXI hardware using the parallel port.
6. The **Communication Path** field should also set itself automatically. It should either be the device number of the DAQ device in the PC that will be used to communicate with the chassis, or it should be the logical device number of the SCXI-1200 module in your chassis, if you have one.
7. After you configure each chassis, press F10 to save the new information. When you have configured all of your chassis, press `Esc` to quit `DAQCONF`. Continue by reading the sections appropriate to your specific programming language and environment later in this chapter for fundamentals on building NI-DAQ applications.

DAQCONF Command-Line Flags

You can use the following command-line flags with DAQCONF:

Command-Line Flag	Description
-t	Disable auto tests
-i	Assume ISA bus computer
-e	Assume EISA bus computer
-a	Auto test for bus type
-u	Usage
-le	Display in English (default)
-lj	Display in Japanese (you must have a Japanese operating system)

You should enter multiple flags separately. For example:

```
daqconf -t -i
```

Using WDAQCONF

WDAQCONF is a Windows-based application that you can use to configure and view National Instruments DAQ device settings for NI-DAQ Windows, LabWindows/CVI, and Windows NT applications (in Windows NT, you must quit all your NI-DAQ applications if you want to configure your DAQ device settings). If you have an EISA computer, you cannot change the device I/O base addresses, interrupt levels, and DMA channels. You must make these changes through the EISA configuration utility described in the *Configuring DAQ Devices on EISA Computers* section earlier in the chapter.

Before using WDAQCONF to configure your device, you must know your device resource settings. You can obtain the information from the DIP switches and jumpers on your devices. If your DAQ devices do not have I/O base address DIP switches or jumpers for interrupt levels and DMA channels, obtain as much information as you can; other resource settings are software programmable. If you have trouble reading DIP

switches or jumper values, consult your DAQ device user manual for assistance.

Locate WDAQCONF in the NI-DAQ Program Group in Windows or Windows NT. Run WDAQCONF by double-clicking on its icon. For Windows NT users, you must terminate all your NI-DAQ applications before you can configure any device. If other NI-DAQ applications are running when you launch WDAQCONF, you can only view your configuration.

When WDAQCONF starts, it tries to retrieve the current configuration from the WDAQCONF.CFG file in the Windows directory (if you are running the Windows NT version of the WDAQCONF, the device configuration is kept in the registry). If WDAQCONF does not locate the file, WDAQCONF will create a file. If WDAQCONF is running on an EISA computer, the utility will retrieve the basic configuration, such as I/O base addresses, interrupt levels, and DMA channels information, from the dedicated system memory. All other information will be obtained from the configuration file.

After WDAQCONF retrieves the current configuration, it displays all the devices installed in a scrollable window. On the right of the window, you see the current setting of the highlighted device.

Perform the following steps to configure, view, and test a DAQ device. Press the F1 key any time to access the online help. If you have an AT-DSP2200, you can follow the same procedure to select a DSP kernel file path for your DSP board.

1. Select a device number for your device by highlighting it in the scrollable window. On the right side of the WDAQCONF window, you can see the current setting of the selected device. The device number you selected is the number you use to refer to the device in your NI-DAQ applications.
2. Click on the **Configure/Test Device #*n*** button to bring out the configuration/test window.
3. Select the **Device** menu item to select your device type. Device type is the name of your device. After you select the device type, you can see the default settings for your device.

☞ **Note:** ***Switchless devices are automatically assigned a device number when*** WDAQCONF ***launches. Therefore, you will not see switchless devices in the*** **Device** ***menu list. Refer to the*** *Configuring Switchless Devices* ***section earlier in this chapter for more information on how*** WDAQCONF ***configures switchless devices.***

4. Click on the DIP switch or select the **DMA** or the **IRQ** menu item to correct any resource setting that does not match your device setting. Select the **Path** menu item to configure the DSP kernel path if you are configuring a DSP board.

   By default, WDAQCONF does not allow you to configure the same resource to different devices. To disable this feature, go to the WDAQCONF main window and uncheck **Resource Checks** under the **Options** menu item.

   You cannot change the **Resource Checks** option if you are currently changing the configuration of any devices, including SCXI. Make sure all your **Device #*n*** windows and your SCXI configuration windows are closed before you try to select this option.

5. To save your device setting, go to the **Configuration** menu item in the configuration window and select save. Before WDAQCONF saves your configuration, WDAQCONF runs through a resource detection test for your configuration. It makes sure you have selected the correct settings. WDAQCONF will not save the configuration if the test fails. You can disable the feature by unchecking the **Auto Test** option under the **Options** menu item in the main window.

   Disabling the **Auto Test** option allows you to run WDAQCONF before installing your device into your computer.

6. After saving your configuration, you can run simple tests on your DAQ device. Under the **Test** menu item in your configuration window, you can see all the tests you can perform.

   - **Configuration** initiates the same test **Auto Test** uses.
   - **Analog I/O** performs single-point analog input read and analog output write operations.
   - **Digital I/O** performs digital input read and digital output write operations.
   - **Counter/Timer** performs simple counter timing.

7. Optionally, you can bring out the hardware window by selecting the **Hardware!** menu item in the configuration window. Notice

that not all National Instruments devices have the **Hardware!** option.

In the hardware window, you can configure other settings such as analog input polarities and analog input mode. If you want to save your new hardware configuration in the hardware window, use the **Save** option under the **Configuration** menu item. If your device has jumpers for setting analog input and analog output parameters and you change any of these settings, you must inform NI-DAQ of these changes either through the hardware configuration window or by calling the `AI_Configure` and `AO_Configure` functions in your program. If you change your hardware configuration here, you are not required to use the `AI_Configure` or `AO_Configure` functions to inform NI-DAQ of your new hardware settings. However, it is still recommended that you do so because these hardware settings may be inadvertently lost if the device configuration is deleted and then reconfigured differently by you or someone else. The same holds true even if your device is software configurable; however, you can skip the step of setting the hardware jumpers on your device.

To close the hardware window, select **Return** under the **Configuration** menu item.

8. After making sure all your DAQ device configurations are correct, you can select **SCXI!** in the main window to configure your SCXI hardware, if any. If you do not have any SCXI hardware to configure, you are ready to begin your NI-DAQ development.

The Resources menu has an option called Write to Text File. Clicking on this option produces a file named `WDAQCONF.TXT` in your Windows directory. This file, which describes your current configuration, is useful when you call National Instruments technical support for assistance.

SCXI Configuration in WDAQCONF

1. Leave the **Chassis ID** set to one. You will use this number to access the SCXI chassis in the NI-DAQ function calls. If you have multiple chassis, advance the **Chassis ID** to configure the next chassis after you have configured the first chassis.
2. Select the correct **Chassis Type** for your chassis; this enables the remaining fields on the panel.
3. If you have only one chassis, leave the **Chassis Address** field and the address jumpers on your SCXI chassis set to zero. If you have

additional chassis, you need to select a unique hardware-jumpered address for each chassis and enter it in the **Chassis Address** field.

4. Leave the **Communication Mode** field alone. This field should set itself automatically to **Serial using DIO port** if you have cabled one of your modules to a DAQ device in the PC. This means that NI-DAQ will use a DIO port of the DAQ device to communicate serially with the SCXI hardware. If you have an SCXI-1200 in your chassis, this field will set itself to be **Parallel using LPT**, meaning that NI-DAQ will communicate with the SCXI hardware using the parallel port.
5. The **Communication Path** field should also set itself automatically. It should either be the device number of the DAQ device in the PC that will be used to communicate with the chassis, or it should be the logical device number of the SCXI-1200 module in your chassis, if you have one.

   If you want to change the **Communication Mode** or **Communication Path** fields, you should do so after you finish configuring all the modules in your chassis in step 6.
6. Enter the configuration for each slot in the chassis. Select the correct **Module Type** that you have installed in the current **Chassis Slot**. The SCXI slots are numbered left to right when you are looking at the front of the chassis, beginning with slot 1. If you did not install a module in this slot, leave this field at **Empty** and advance the **Chassis Slot** number to the next slot.

   Press the **Configure Module *n*** button to bring up the Module Configuration window, and set the following fields:

   a. If the module in this slot is not an SCXI-1200, the next field is **Cabled Device**. If the module in this slot is *directly* cabled to a DAQ device in your computer, set this field to the **Device Number** for that device. Leave the **Cabled Device** field at **None** if the module in the current slot is not directly cabled to a DAQ device in your PC.

      If you are operating your modules in Multiplexed mode, you only need to cable one module in each chassis to your plug-in DAQ device in the PC. If you are using an SCXI-1200 module in your chassis, you do not need to cable any modules in your chassis to a DAQ device in the PC; the SCXI-1200 can control the chassis.

   b. **Operating Mode**. Multiplexed mode is the default operating mode—it is recommended for almost all SCXI applications.

The operating modes available for each SCXI module type are discussed in *The SCXI Hardware* section of Chapter 2, *Hardware Overview*.

For the SCXI-1200, the **SCXI Multiplexed** option is analogous to the Multiplexed mode. If you select the **Standalone** mode, you disable the SCXI-1200 from communicating with other modules in the chassis.

c. If the module in this slot is an SCXI-1200, you must also configure the following fields:

Logical Device Number. You must assign the SCXI-1200 a logical device number so that you can use the SCXI-1200 with the NI-DAQ functions as if it were a plug-in DAQ device in the PC. The SCXI-1200 has the same functionality as the Lab-PC+ plug-in board. After you finish your SCXI configuration, the SCXI-1200 will be configured for this **Logical Device Number** in the device configuration panel.

Connected to… is the parallel port address to which you have cabled the SCXI-1200.

IRQ Level is the interrupt level you want the parallel port interrupt service routines to use. During a data acquisition or waveform generation on the SCXI-1200, NI-DAQ services parallel port interrupts to transfer data between the SCXI-1200 and your PC memory. This IRQ level must correspond to the IRQ level used by the parallel port to which the SCXI-1200 is connected.

d. The remaining fields in the Module Configuration window (**Terminal Block** and **Channel Configuration** settings) are not used by NI-DAQ, they are only used by LabVIEW. However, you can enter the correct information here for your own use and easy lookup.

e. Press the **Save** button to save this module configuration, and press the **Return** button to return to the SCXI Configuration window to configure the next module.

7. After you configure each chassis, select **Save** from the **Configuration** menu to save the new information. When you have configured all of your chassis, close the SCXI Configuration window and exit `WDAQCONF`. Continue by reading the sections appropriate to your specific programming language and environment later in this chapter for fundamentals on building NI-DAQ applications.

☞ **Note:** ***You can select* Test *from the* Configuration *menu to test your entire SCXI configuration.***

The Fundamentals of Building DOS Applications with NI-DAQ

This section contains general information about building NI-DAQ applications that run in DOS and explains the nature of the files needed and the basics of making applications using the following compilers:

- Microsoft C
- Microsoft Visual Basic
- Turbo C++ and Borland C++
- Borland Turbo Pascal

In the DOS environment, a set of function libraries provides the NI-DAQ functions. You compile and then link an application that makes calls to these functions to the appropriate library for that compiler.

Creating a DOS Application Using Microsoft C

The NI-DAQ library for Microsoft C (`NIDAQMSC.LIB`) is compiled using the large memory model. It is therefore essential that you install the large memory model of your Microsoft C Compiler. Perform the following steps:

1. Create your source code. Follow the instructions in this manual and the *NI-DAQ Function Reference Manual for PC Compatibles* when making calls to NI-DAQ functions. Be sure to use the functional prototypes by including `NIDAQ.H` in your source file. You can find the file `NI_DAQ.H` in the `C_EX` subdirectory under your NI-DAQ directory.

☞ **Note:** ***You must call one of the* USE *functions in your application before calling any other NI-DAQ functions. These functions cause portions of the NI-DAQ library that are required to use your DAQ product to be included in your application. If you do not call the appropriate* USE *function, your other NI-DAQ functions will return* error -421 *(*functionNotLinkedErr*). If you are using a Lab-PC+ for example, you must call the* USE_LAB *function before calling any other NI-DAQ function.***

2. Compile your source code with the Microsoft C Compiler (Version 8.0 or later) and use the large memory model, which you select when you include the /AL flag in the command line. For example, to compile aiaoxmpl.c and its support files, use the following commands:

   ```
   cl /c /AL aiaoxmpl.c
   cl /c /AL getdev.c
   cl /c /AL errprint.c
   ```

   The /c flag directs the compiler to compile only.

3. Link your object file or files (using Microsoft Overlay Linker Version 3.61 or later) with the NIDAQMSC.LIB library to create the executable application. For example, to link the aiaoxmpl.obj, getdev.obj, and errprint.obj files produced in step 2, use the following command:

   ```
   link /SEG:250 aiaoxmpl getdev errprint,,,NIDAQMSC;
   ```

   This link command will produce an aiaoxmpl.exe executable.

   The /SEG:250 flag is necessary if you are using an E Series device and one or more of the USE_E_Series functions. By default, the Microsoft Overlay linker sets the maximum number of program segments to 128. The /SEG:250 flag increases the maximum number of segments to 250.

   The /ST flag is needed to increase the stack size from its default value of 2,048 bytes. Otherwise, the program fails with a stack-overflow message.

Example Programs

You can find a set of example programs and the necessary header files in the NIDAQDOS\C_EX directory.

Creating a DOS Application Using Visual Basic

To create an application that calls NI-DAQ functions, first create a source file for your application using the following guidelines:

1. Add the following line to the beginning of the source file:

   ```
   REM $INCLUDE: 'NIDAQ.INC'
   ```

   This statement declares all of the NI-DAQ functions in the NI-DAQ library.

☞ **Note:** ***If you are using NI-DAQ memory management functions, use the include file called*** `NIDAQR.INC`***, which has less restrictive prototypes than*** `NIDAQ.INC`***. Do NOT include both*** `NIDAQ.INC` ***and*** `NIDAQR.INC` ***in the same source file.***

2. NI-DAQ library needs to allocate some memory for internal use. Therefore, you need to set aside memory using the `SETMEM` statement. The amount of memory you need will depend on which NI-DAQ functions you are using. If you have not set aside sufficient memory, NI-DAQ functions will return a memory error (error code -98). For a description of the `SETMEM` statement, refer to your BASIC manual. In the NI-DAQ Basic example programs, a number between -2,000 and -10,000 is generally used as follows:

   ```
   heap.size=SETMEM (-8000)
   ```

3. Follow the instructions in this manual and the *NI-DAQ Function Reference Manual for PC Compatibles* when making calls to the NI-DAQ functions. Remember to substitute a period (.) wherever you see an underscore (_) in a function name. For example, the function `AI_Configure` should be entered as `AI.Configure` in Visual Basic applications.

☞ **Note:** ***You must call one of the*** `USE` ***functions in your application first, before calling any other NI-DAQ functions. These functions cause portions of the NI-DAQ library that are required to use your DAQ product to be included in your application. If you fail to call the appropriate*** `USE` ***function, your other NI-DAQ functions will return error -421 (*****functionNotLinkedErr*****). If you are using a Lab-PC+ for example, you must call the*** `USE.LAB` ***function first, before calling any other NI-DAQ function.***

Next, you can use either of the following approaches to run your application:

1. Run your application inside the Visual Basic environment. To do so, you must first create and then load a Quick library of NI-DAQ functions when you enter Visual Basic environment. The only case in which this approach will not work is when Visual Basic returns out-of-memory error; in that case, use the second approach.
2. Compile and run your application from the DOS prompt. To do so, use the BASIC command-line compiler and linker.

These approaches are explained in detail in the following sections.

Running Your Application Inside the Visual Basic Environment

First, you must create an NI-DAQ Quick library.

MAKEQLB.BAT in the QLBUTIL subdirectory is useful for creating Quick libraries for Visual Basic.

The steps for making a Quick library are as follows:

1. Edit NIDAQ.BAS. Remove the keyword REM from functions you want to include in the Quick library.
2. Run this batch file by using the following command:

   ```
   MAKEQLB VB
   ```

3. If all files needed to build the Quick library are found, and the linking was successful, the batch file creates a Quick library in the NI-DAQ LIB subdirectory with a .QLB extension.

Next, load the Quick library when you enter the environment by using the following command:

```
vbdos /l NIDAQVB
```

☞ **Note:** ***Visual Basic returns an out-of-memory error either when you try to load the Quick library or when you try to run your application. You may try to free up memory by removing as many TSRs or device drivers as possible before entering the Visual Basic environment.***

After you are inside the environment, you can load the source file of your application and run it.

You can substantially reduce the size of the Quick library by including only those USE functions that apply to your product. The addition of one or more USE functions will increase the size of your Quick library. As a result, you may not have enough memory to load the Quick library into the Visual Basic environment.

Compiling and Running Your BASIC Application from the DOS Prompt

The steps to run your application outside BASIC environment are as follows:

1. Compile your source code with the Visual Basic compiler. For example:

   ```
   bc /O aiaoxmpl.bas;
   ```

☞ **Note:** ***NOT ENOUGH MEMORY—If the Visual Basic compiler does not have enough memory to compile your application, you should first try to make available as much conventional memory as possible. See your DOS manual for information on how to do so. If you still cannot compile your application, you can edit the files*** `NIDAQ.INC` ***(or*** `NIDAQR.INC`***) and*** `NIDAQCNS.INC` ***to reduce their size.***

2. Link the object file (using Microsoft Overlay Linker Version 3.61 or later) produced in step 1 with `NIDAQMSC.LIB`, `SUP71.LIB`, and the Visual Basic library. For example:

```
link/NOE/NOD /SEG:200 aiaoxmpl,,, VBDCL10E NIDAQMSC SUP71; (MS-DOS)
link/NOE/NOD /SEG:200 aiaoxmpl,,, VBDCL10 NIDAQMSC SUP71; (NEC-DOS)
```

The `/SEG:200` flag is necessary if you are using several of the `USE` functions. By default, the Microsoft Overlay linker sets the maximum number of program segments to 128. The `/SEG:200` flag increases the maximum number of segments to 200. Most of the examples will link successfully with a smaller value for the `SEG` flag.

You must include the `SUP71` library in your link command because the DOS NI-DAQ library `NIDAQMSC.LIB` is compiled using the Microsoft C compiler. Microsoft C support functions are contained in the `SUP71` library, which you can find in the NI-DAQ `LIB` subdirectory. You should include the NI-DAQ `LIB` directory and the Visual Basic `LIB` directory in your `LIB` environment variable so the linker can find the libraries. The following statement is an example of how to set the `LIB` environment variable in your `autoexec.bat` file:

```
SET LIB=C:\NIDAQDOS\LIB;C:\VBDOS\LIB
```

Example Programs

You can find a set of example programs and the necessary header files in the `NIDAQDOS\BASIC_EX` directory.

Creating a DOS Application Using Borland Turbo C++ or Borland C++

The NI-DAQ libraries for Borland Turbo C++ and Borland C++ are compiled using the large memory model. Therefore, it is essential that you install the large memory model of your C++ compiler.

To create your application that calls NI-DAQ functions, first create source code. Follow the instructions in this manual and the *NI-DAQ*

Function Reference Manual for PC Compatibles when making NI-DAQ function calls. Be sure to use the function prototypes by including NIDAQ.H in your source file.

☞ **Note:** ***You must call one of the*** USE ***functions in your application before calling any other NI-DAQ functions. These functions cause portions of the NI-DAQ library that are required to use your DAQ product to be included in your application. If you do not call the appropriate*** USE ***function, your other NI-DAQ functions will return error -421 (*****functionNotLinkedErr*****). If you are using a Lab-PC+ for example, you must call the*** USE_LAB ***function before calling any other NI-DAQ function.***

To compile and run your application, it is recommended that you use the Integrated Development Environment (IDE). You can find example project files created in version 3.1 in the NI-DAQ C_EX directory. Newer versions of Borland C++ can use 3.1 project files.

To run your application using the IDE, you must follow these guidelines:

1. Open a project to manage your application code. Include the NI-DAQ library NIDAQBC.LIB along with the source file in your project.
2. Choose **Options | Compiler | Code Generation** from the main menu. A dialog box will be displayed. Set the **Model** button to **Large**. This will direct the compiler to use the large memory model.
3. Choose **Options | Directories** from the main menu. A dialog box will be displayed that lets you set the path for include files, libraries, and so on. Add the path to NI-DAQ LIB subdirectory. Also add the path to the NI-DAQ C_EX directory to the include path.

If you are using Borland C++ version 4.0, you need to edit the definition of halloc in the include file MALLOC.H in the include directory. To do so, go to line 65 in MALLOC.H and remove the underscore from in front of farmalloc. The line should then read as follows:

```
#define halloc(num, size) (void huge *)farmalloc((unsigned long)(num)*
(size))
```

This change correctly maps the halloc function, which several NI-DAQ for DOS example programs use.

Example Programs

You can find a set of example programs and the necessary header files in the NIDAQDOS\C_EX directory. Not all of the examples will work with Borland C because some of them use plotting routines written with Microsoft C. However, you can comment out the plotting operations and use those example programs.

Creating a DOS Application Using Borland Turbo Pascal

To create a Turbo Pascal application that calls NI-DAQ functions, perform the following steps:

1. Create your source code. Follow the instructions in this manual and the *NI-DAQ Function Reference Manual for PC Compatibles* when making NI-DAQ function calls.

☞ **Note:** ***You must call one of the*** USE ***functions in your application before calling any other NI-DAQ functions. These functions cause portions of the NI-DAQ library that are required to use your DAQ product to be included in your application. If you do not call the appropriate*** USE ***function, your other NI-DAQ functions will return error -421 (*****functionNotLinkedErr*****). If you are using a Lab-PC+ for example, you must call the*** USE_LAB ***function before calling any other NI-DAQ function.***

2. Add the NIDAQ unit to the USES clause in your source code.
3. When compiling your program, be sure that the Turbo Pascal compiler can locate the NI-DAQ units.
   - If you are using the Integrated Development Environment (IDE), the directory containing the NI-DAQ units should be specified as one of the Unit directories under the **Options|Directories** menu.

     We recommend that you choose Auto Save for the Environment in the **Options|Environment|Preferences** menu to make the change permanent.
   - If you are using the command-line version of the compiler, specify the directory using the /Uxxx switch.
   - You may notice the compiler directive {$N+} in some of the NI-DAQ Turbo Pascal example programs. This option directs the compiler to generate inline 80*x*87 code for handling floating point numbers. This code is required by NI-DAQ routines that use variables of type *Double*.

Memory Requirement

If the Turbo Pascal interactive environment runs out of memory while compiling NI-DAQ applications, try some or all of the following:

- Set the Turbo Pascal compile destination to disk.
- Set the Turbo Pascal link buffer to disk.
- Remove as many TSRs or device drivers as possible before compiling.
- It is sometimes possible to compile with less memory if, instead of specifying the main NI-DAQ Turbo Pascal Unit (TPU), you specify only the needed sub-TPUs. To do this, replace the `USES NIDAQ;` statement in your application with `USES xxx;` where *xxx* is one or more of the sub-TPUs containing the function or functions you want. The `TPULIST.TXT` file in the NI-DAQ `PAS_EX` directory contains a table that lists the TPUs and the functions they contain.

Using these units instead of using `nidaq.tpu` prevents Turbo Pascal from having to bring all the units into memory. Notice that the final executable file is only slightly smaller than a program using `nidaq.tpu` because Turbo Pascal does not include any unused code in the final executable file.

If you are not running out of memory during compilation (such as when using the TPCX to compile), this method of bypassing `nidaq.tpu` may not offer you any significant advantage.

Example Programs

You can find a set of example programs and the necessary header files in the `NIDAQDOS\PAS_EX` directory.

The Fundamentals of Building Windows Applications with NI-DAQ

This section contains general information about building NI-DAQ applications, describes the nature of the NI-DAQ files used in building NI-DAQ applications, and explains the basics of making applications using the following tools:

- Borland C++ for Windows
- Microsoft Visual C++

- Borland Turbo Pascal for Windows
- Microsoft Visual Basic

If you are not using the tools listed, consult your development tool reference manual for details on creating applications that call DLLs.

The NI-DAQ Libraries

The NI-DAQ for Windows function libraries are DLLs, which means that NI-DAQ routines are not linked into the executable files of applications. Only the information about the NI-DAQ routines in NI-DAQ import libraries are stored in the executable files. For that reason, Windows-executable files are usually smaller than DOS-executable files.

☞ **Note:** ***FOR USERS OF NI-DAQ 4.5.2 AND EARLIER ONLY—The NI-DAQ DLL is*** `NIDAQ.DLL`***. The NI-DAQ import library is*** `NIDAQ.LIB`***. In NI-DAQ versions prior to version 4.6, the DLL was called*** `ATWDAQ.DLL` ***and the import library was*** `ATWDAQ.LIB`***. If you recompile and link an old program, you need to use the name*** `NIDAQ` ***instead of*** `ATWDAQ`***.***

Import libraries contain information about their DLL-exported functions. They indicate the presence and location of the DLL routines. Depending on the development tools you are using, you may give the DLL routines information through import libraries or through function declarations.

Using functional prototypes is a good programming practice. That is why NI-DAQ is packaged with functional prototype files for four different Windows development tools. The installation utility copies the appropriate prototype files for the development tools you choose. If you are not using any of the four development tools that NI-DAQ supports, you must create your own functional prototype file.

Calling Windows DLL functions with their ordinal numbers defined improves Windows performance. For this reason, National Instruments has documented the ordinal numbers for all the exported functions in a file named `WDAQFUNC.ORD`.

NI-DAQ Programming Considerations

In addition to knowing how to use the NI-DAQ DLL, you should consider some special problems that can occur when you access certain NI-DAQ routines. This section briefly describes the nature of the

problems. The following sections, which are specific to each language, give the methods for solving the problems.

Buffer Allocation

Allocating memory in a Windows application is much more restrictive than is normally encountered in a non-Windows application. Windows requires you to allocate all memory through the Windows memory manager, and thus has its own memory-allocation functions. In most cases, you should use these functions rather than the memory-allocation functions normally used by a specific language.

Huge (Greater Than 64 KB) Buffer Access

Buffers of allocated memory that exceed 64 KB are divided into 64 KB groups, or *segments*. When you are accessing data within the buffer and you reach the end of one of these segments and must reference the next segment, you need some way of finding the address of the next segment. This event is called *crossing a segment boundary*. Some languages have special types of pointers that make this crossing transparent to the programmer; other languages require you to perform your own pointer arithmetic using a Windows-supplied constant to increment your pointer address.

String Passing

When NI-DAQ for Windows routines call for a string that is passed as a parameter, the routines expect a pointer to a null-terminated string. Some languages require special string handling to support this type.

Parameter Passing

You can pass procedure or function parameters by value or by reference. Different languages have different default settings. You must be sure to pass certain variables by value or by reference to each NI-DAQ for Windows function.

Data Acquisition with DMA

In NI-DAQ for Windows, it is possible to use data acquisition buffers above 16 MB if you are using `NIVISRD.386`, which is the default option. NI-DAQ copies data from a buffer below 16 MB to or from your data acquisition buffer above 16 MB.

If you do not use NIVISRD.386, or you are using NI-DAQ for DOS, the DMA region is limited to the first 16 MB of physical memory because of the AT and NEC bus architectures. Therefore, if any portion of a data acquisition DMA buffer is mapped above the first 16 MB of memory, NI-DAQ returns the **invalidMemRegionErr** error (-199).

To avoid this limitation, use Set_DAQ_Device_Info to switch data acquisition to interrupt-driven mode or use an EISA computer.

Creating a Windows Application Using Borland C++

This section assumes that you will be using the Borland IDE to manage your code development.

For Windows programs in general, remember to follow this procedure:

1. Open a project module to manage your application code.
2. Create files of type .cpp (C++ source code).
3. Set Options\Application to Windows App to set options similar to those used in a module definition file.
4. Create your resources using the Borland Whitewater Resource Toolkit. After you have created the resources, save them into a .res file and add the .res file to the list of files for the project window.

To use the NI-DAQ functions, you must use the NI-DAQ DLL. Follow this procedure:

1. Create your source file as you would for other Windows programs written in C++, calling NI-DAQ functions as typical function calls.
2. Prototype any NI-DAQ routines used in your application. Include the NI-DAQ header file, which prototypes all NI-DAQ routines, as shown in the following example:

   ```
   #include "WDAQ_BC.H"
   ```

3. Add the NI-DAQ import library NIDAQ.LIB to the project module.

Example Programs

You can find some example programs and project files created in version 3.1 in the NIDAQWIN\BCPP_EX directory. Newer versions can use 3.1 project files.

Special Considerations

Buffer Allocation

To allocate memory, you can use the Windows functions `GlobalAlloc()` and `GlobalFree()` or an NI-DAQ memory management function, `NI_DAQ_Mem_Alloc` or `NI_DAQ_Mem_Free`. After allocation, to use a buffer of memory, you must lock memory with `GlobalLock()` or `NI_DAQ_Mem_Lock`. After using the memory, you must unlock memory with `GlobalUnlock()` or `NI_DAQ_Mem_Unlock`.

☞ **Note:** ***If you allocate memory from*** `GlobalAlloc()`***, call*** `GlobalLock()` ***and*** `GlobalPageLock()` ***on the memory object before passing it to NI-DAQ.***

Huge Buffer Access

When referencing memory buffers that may exceed 64 KB in size, use *huge* pointers to reference the buffer. Any other pointer type will not perform the correct pointer increment when crossing the 64 KB segment boundary. When you use the *huge* pointer, C automatically adjusts for segment wraparound and normalizes the segment for pointer comparison.

String Passing

To pass strings, pass a pointer to the first element of the character array. Be sure that the string is null-terminated.

Parameter Passing

By default, C passes parameters by value. Remember to pass pointers to the address of a variable when you need to pass by reference.

Creating a Windows Application Using Microsoft Visual C++

This section assumes that you will be using the Microsoft Visual Workbench to manage your code development.

For Windows programs in general, remember to follow this procedure:

1. Open a project module to manage your application code.
2. Create files of type `.cpp` (C++ source code).
3. Create a module definition file, and add it to the project.

4. Create your resources using the App Studio. After you have created the resources, save them into an `.rc` file and add the `.rc` file to the project.

To use the NI-DAQ functions, you must use the NI-DAQ DLL. Follow this procedure:

1. Create your source file as you would for other Windows programs written in C++, calling NI-DAQ functions as typical function calls.
2. Prototype any NI-DAQ routines used in your application. Include the NI-DAQ header file, which prototypes all NI-DAQ routines, as shown in the following example:

   ```
   #include "WDAQ_C.H"
   ```

3. Add the NI-DAQ import library `NIDAQ.LIB` to the project module.

Special Considerations

See *Special Considerations* in the *Creating a Windows Application Using Borland C++* section earlier in this chapter.

Creating a Windows Application Using Turbo Pascal

For Windows programs in general, remember the following points:

1. Turbo Pascal for Windows 1.0 and 1.5 users: Create files of type `.pas` (Pascal source code), including the Windows object units `WObjects`, `WinTypes`, and `WinProcs`.

   Turbo Pascal 7.0 users: Create files of type `.pas` (Pascal source code), including the units `OWindows`, `ODialogs`, `WinTypes`, and `WinProcs`.

2. Create your resources using the Borland Whitewater Resource Toolkit and save the resources into a `.res` file. You must add this resource to the executable file by using the `{$R ...}` compiler command.
3. Turn on the `{$N+}` compiler option to enable the extended floating-point types. You can use this option whether or not you actually have a math coprocessor; if you do not have a coprocessor, Turbo Pascal will emulate one for you. NI-DAQ functions expect to receive 8-byte floating-point values; with the `$N` option enabled, Turbo Pascal for Windows can generate an 8-byte variable of type `double`. Otherwise, with this option

disabled, Turbo Pascal can only generate a 6-byte `real`, which is not compatible with NI-DAQ routines.

To use the NI-DAQ functions, you must use the NI-DAQ DLL. You will *not* be using the import library (as in C or C++) to reference the DLL, however. Follow this procedure:

1. Create your source file as you would for any other Windows program written in Pascal, calling NI-DAQ functions as typical function calls.
2. Prototype any NI-DAQ routines used in your application. Include the NI-DAQ include file, which prototypes all NI-DAQ routines, as shown in the following example:

```
{$I WDAQ_TP.INC}
```

☞ **Note:** ***This include file defines a special pointer to a double type called*** `PDouble`***. Use*** `PDouble` ***in a manner similar to that of the Turbo Pascal for Windows type*** `PInteger`***.***

Example Programs

You can find a set of example programs and the necessary header files in the `NIDAQWIN\TP_EX` directory.

Special Considerations

Buffer Allocation

To allocate memory, you can use the Windows functions `GlobalAlloc()` and `GlobalFree()` or an NI-DAQ memory management function, `NI_DAQ_Mem_Alloc` or `NI_DAQ_Mem_Free`. After allocation, to use a buffer of memory, you must lock memory with `GlobalLock()` or `NI_DAQ_Mem_Lock`. After using the memory, you must unlock memory with `GlobalUnlock()` or `NI_DAQ_Mem_Unlock`.

☞ **Note:** ***If you allocate memory from*** `GlobalAlloc()`***, call*** `GlobalLock()`***, and*** `GlobalPageLock()` ***on the memory object before passing it to NI-DAQ.***

Huge Buffer Access

Unlike C and C++, Turbo Pascal does not support *huge* pointers. Consequently, you must perform your own pointer arithmetic when accessing memory buffers greater than 64 KB in size. Essentially,

whenever you increment a pointer to a buffer of memory, you should check the low word of the pointer to see if it *rolls over* from $FFFF back to $0000. In this case, you need to increment the high word of the pointer by a value given as `Ofs(AHIncr)`. This increments the Windows selector by the correct amount and references the next 64 KB segment. By using record variants like `PMemory` used in `DAQOP_TP.PAS`, you can easily access both the pointer and the high and low words of the pointer value. For more details, please see your Turbo Pascal manuals.

String Passing

Normally, standard Pascal strings consist of an array of up to 255 characters, with the first byte reserved for the length of the existing string. However, Windows and NI-DAQ functions expect a null-terminated string, such as those used in the C language. Fortunately, Turbo Pascal for Windows extends the string syntax to support the null-terminated string. To use this option, check to ensure that the extended syntax compiler option `{$X+}` is enabled (which is the default), and then declare the string as an array of characters, as in the following example:

```
type
    Tfilename = array[0..80] of Char;
begin
    err := DAQ_to_Disk(.., Tfilename, ...);
```

In addition, Turbo Pascal has a predefined pointer to a null-terminated string called `PChar`. To pass a null-terminated string to a procedure or function, pass either a `PChar` pointer variable to the string, or pass the name itself without an index.

Parameter Passing

By default, Pascal passes parameters by value. Include the `var` keyword if you need to pass by reference.

☞ **Note:** ***Functions such as*** `DAQ_Monitor` ***or*** `Align_DMA_Buffer` ***return variables (***`newestPtIndex` ***and*** `AlignIndex`***) that index certain buffers. These values assume that the index of your first index is zero. If your Pascal array starts at one, you must add one to these variables if you use them.***

Creating a Windows Application Using Microsoft Visual Basic

To use the NI-DAQ functions, you must use the NI-DAQ DLL. Follow this procedure:

1. Create your forms and code as you would for any other Visual Basic program, calling NI-DAQ functions as typical function calls.
2. Prototype any NI-DAQ routines used in your application. You can do this by adding the NI-DAQ header module `WDAQ_VB.BAS` in the NI-DAQ `VB_EX` directory. Go to the **File** menu and select the **Add File** option. Then, using the file dialog box, find `WDAQ_VB.BAS` and click on the **OK** button. Verify the file's existence in the *project* window. This header file will prototype all NI-DAQ functions.

☞ **Note:** ***Use*** `WDAQR_VB.BAS` ***if you are using NI-DAQ memory management functions. Do NOT add both*** `WDAQ_VB.BAS` ***and*** `WDAQR_VB.BAS` ***to the same project.***

In Visual Basic, function declarations have scope globally throughout the project. In other words, you can define your prototypes in any module. The functions will be recognized even in other modules.

For information on using the NI-DAQ Visual Basic Custom Controls, see the *NI-DAQ Events in Visual Basic for Windows* ***section in Chapter 3,*** *Software Overview.*

Please also refer to the *Programming Languages Considerations* ***section in Chapter 1,*** *Using the NI-DAQ Functions****, of the*** *NI-DAQ Function Reference Manual for PC Compatibles* ***for more information on using the NI-DAQ functions in Visual Basic for Windows.***

Example Programs

You can find a set of example programs and the necessary header files in the `NIDAQWIN\VB_EX` directory.

Special Considerations

Buffer Allocation

Visual Basic is quite restrictive when allocating memory. You allocate memory by declaring an array of whatever data type with which you want to work. Visual Basic supports dynamic memory allocation by

allowing you to redimension an array to a variable size during run-time. However, arrays are restricted to being less than 64 KB in *total* size (this translates to about 32,767 integers, 16,384 long integers, or 8,191 doubles). To break the 64 KB buffer size barrier, you can use NI-DAQ memory management functions, with which you can use buffers larger than 64 KB. For more information on NI-DAQ memory management functions, see *The Memory Management Functions* section in Chapter 3, *Software Overview*.

Pay special attention to NI-DAQ routines that modify string buffers, such as the `DAQ_DB_StrTransfer` routine. You must ensure that the memory buffer is already allocated to a size large enough to accommodate all of the requested samples. The following example code copies a string buffer to disk:

```
:
open "filename.dat" for Binary As fh%
strBuffer$ = String$ (numSamples, 0)'Allocate space for half buff
:
daqErr% = DAQ_DB_StrTransfer (board%, strBuffer$, ptsTfr&, status%)
Put fh%, , strBuffer$
:
close fh%
:
```

Huge Buffer Access

Visual Basic does not support buffer allocation greater than 64 KB or huge buffer access. To allocate and use buffers that are larger than 64 KB, consult the NI-DAQ memory management functions.

String Passing

In Visual Basic, variables of data type `String` need no special modifications to be passed to NI-DAQ for Windows functions. Visual Basic automatically appends a null character to the end of a string before passing it (by reference, because strings cannot be passed by value in Visual Basic) to a procedure or function.

Parameter Passing

By default, Visual Basic passes parameters by reference. Include the `ByVal` keyword if you need to pass by value.

The Fundamentals of Building Windows NT Applications with NI-DAQ

This section contains general information about building NI-DAQ applications, describes the nature of the NI-DAQ files used in building NI-DAQ applications, and explains the basics of making NI-DAQ applications using the following tool:

- Microsoft Visual C++ for Windows NT Version 1.0

If you are not using the tool listed, consult your development tool reference manual for details on creating applications that call the Windows NT DLL.

The NI-DAQ for Windows NT Files

The NI-DAQ for Windows NT function library includes a DLL and two Windows NT kernel device drivers, `NIDAQNT.SYS` and `NIDAQKNL.SYS`. Most of the functionality is built into the kernel mode device driver, `NIDAQNT.SYS`.

`NIDAQKNL.SYS` is installed only on ISA-bus computers and helps users do DMA transfers into physical memory about 16 MB. The DLL provides a portable programming interface for NI-DAQ applications.

The NI-DAQ for Windows NT DLL, `NIDAQ.DLL`, is located in your Windows NT `SYSTEM32` directory. You can find the kernel mode device drivers in your Windows NT `SYSTEM32\DRIVERS` directory.

In addition to the system files, NI-DAQ for Windows NT is shipped with other files to help you develop NI-DAQ applications. These files are in the directory you select when installing NI-DAQ for Windows NT.

- `WDAQ_C.H` is an include file that contains all NI-DAQ function prototypes. You should include this file in your source files when you build your NI-DAQ applications.
- `NIDAQ.LIB` is an import library for the `NI-DAQ DLL`. When linking NI-DAQ applications, include this library in your link command in order to resolve all your NI-DAQ function calls.

Creating a Windows NT Application Using Microsoft Visual C++ for Windows NT

The best way to learn how to build an NI-DAQ application is to study the example programs NI-DAQ provides. AIAOXMPL.C is the simplest example program for Windows NT. The make files shipped with the examples are generated by the Visual C++ IDE. You can use them as project files in the Visual C++ environment or use them with NMAKE.EXE for the Visual C++ command-line compiler. Carefully study the make file on how to build AIAOXMPL.EXE.

Before you run the NMAKE command, be sure of the following:

1. Your INCLUDE environment variable is pointing to the Visual C++ include directory.
2. The environment variable CPU is defined as i386.
3. The compiler can locate WDAQ_C.H when it compiles the example program. The NI-DAQ install utility puts the file in the NI-DAQ example program directory. If you compile unmodified NI-DAQ example programs, this will not be a problem.
4. The linker can locate NIDAQ.LIB when it links the example program. The NI-DAQ install utility puts the file in the NI-DAQ library directory. If you use the NI-DAQ unmodified example make file, this will not be a problem.

When you are ready to make the example program, switch to the NI-DAQ example directory and type the following:

```
nmake /f aiaoxmpl.mak
```

to make AIAOXMPL.EXE.

The NI-DAQ example make files use most of the default compiler and link options. If your NI-DAQ applications demand other options, consult your compiler manuals for more information.

Special Considerations

Changing I/O Page Lock Limit

Windows NT limits the total amount of memory that is nonswappable (page locked). The default amount of memory is determined according to the total amount of physical memory available in your system. If you are working with large data acquisition buffers, you can increase the default I/O page lock limit in WDAQCONF.EXE. To change the page

lock limit, select **IOPageLockLimit** under the **Options** menu and enter a new page lock limit. Your changes will go into effect when you reboot and restart Windows NT.

User Privilege Level When Using NI-DAQ

Windows NT provides different security levels for different kinds of users. Depending on your privilege level, you have limitations on what you can do with NI-DAQ.

The NI-DAQ kernel mode device driver `NIDAQKNL.SYS` is loaded at system boot time and is never unloaded from the system.

By default, NI-DAQ applications load the NI-DAQ kernel mode device driver `NIDAQNT.SYS` on demand. Windows NT loads the NI-DAQ device driver `NIDAQNT.SYS` when you run your first NI-DAQ application. Windows NT unloads the driver `NIDAQNT.SYS` when you terminate the last NI-DAQ application. This default behavior saves memory when you are not running an NI-DAQ application. Loading and unloading device drivers require the highest privilege level, administrator level.

However, you may not want to always run your NI-DAQ application in the highest privilege level. To change the default behavior, you need to change the load option for NI-DAQ driver `NIDAQNT.SYS` in the **Control Panel** as follows:

1. Go to the **Control Panel** and select **Devices**.
2. Use the scroll bar and scroll down the driver list until you see **NIDAQNT**.
3. Highlight **NIDAQNT** by clicking on it once.
4. Click on the **Startup** button on the right.
5. Change the startup type to **Automatic** and press **OK**.
6. Close the **Devices** window and **Control Panel**.

The next time you reboot and restart Windows NT, Windows NT will automatically load the NI-DAQ driver `NIDAQNT.SYS` when it boots. You can then log in as any type of user and run NI-DAQ applications.

However, there is one drawback to this option. `WDAQCONF` needs to be able to load and unload the NI-DAQ device driver `NIDAQNT.SYS` on the fly in order to configure DAQ devices. After you change the driver load option to automatic, you will not be able to configure any DAQ

devices through WDAQCONF. You can run WDAQCONF only in read-only mode. If you need to configure a DAQ device, you must change the load option back to manual and reboot Windows NT.

Hardware Overview

Chapter 2

This chapter contains hardware information concerning your National Instruments DAQ device.

The MIO and AI Multifunction I/O Devices

The National Instruments MIO and AI devices differ in some respects depending on the timing and control hardware on them. We will refer to the AT-MIO-16, AT-MIO-16D, AT-MIO-16F-5, AT-MIO-16X, and AT-MIO-64F-5 boards as Am9513-based devices because they contain the Am9513 System Timing Controller chip. Timing and control on the E Series devices is performed by the National Instruments Data Acquisition System Timing Controller (DAQ-STC) ASIC.

MIO and AI Device Analog Input

Table 2-1 summarizes the key analog input characteristics for the MIO and AI multifunction I/O devices.

Table 2-1. MIO and AI Multifunction I/O Device Analog Input Characteristics

Device	Number of Channels	ADC Resolution (Bits)	Gains	Range (V)	Input FIFO (Words)	Hardware Analog Trigger	Fully Software Configurable
AT-MIO-16E-2, NEC-MIO-16E-4, NEC-AI-16E-4	16	12	0.5, 1, 2, 5, 10, 20, 50, 100	±5, 0 to 10	2,048	yes	yes
AT-MIO-64E-3	64	12	0.5, 1, 2, 5, 10, 20, 50, 100	±5, 0 to 10	2,048	yes	yes
AT-MIO-16E-10, AT-MIO-16DE-10	16	12	0.5, 1, 2, 5, 10, 20, 50, 100	±5, 0 to 10	512	no	yes
AT-MIO-16E-1	16	12	0.5, 1, 2, 5, 10, 20, 50, 100	±5, 0 to 10	8,192	yes	yes

This document was created with FrameMaker 4.0.4

Table 2-1. MIO and AI Multifunction I/O Device Analog Input Characteristics (Continued)

Device	Number of Channels	ADC Resolution (Bits)	Gains	Range (V)	Input FIFO (Words)	Hardware Analog Trigger	Fully Software Configurable
AT-MIO-16XE-50, NEC-MIO-16XE-50, NEC-AI-16XE-50, DAQPad-MIO-16XE-50	16	16	1, 2, 10, 100	±10, 0 to 10	512	no	yes
AT-MIO-16F-5	16	12	0.5, 1, 2, 5, 10, 20, 50, 100	0 to +10, ±5	256	no	yes
AT-MIO-64F-5	64	12	0.5, 1, 2, 5, 10, 20, 50, 100	0 to +10, ±5	512	no	yes
AT-MIO-16X	16	16	1, 2, 5, 10, 20, 50, 100	0 to +10, ±10	512	no	yes
AT-MIO-16L, AT-MIO-16DL	16	12	1, 10, 100, 500	0 to +10, ±5,	16 (512 on DL)	no	no
AT-MIO-16H, AT-MIO-16DH	16	12	1, 2 4, 8	0 to +10, ±5,	16 (512 on DH)	no	no

☞ **Note:** ***Terms such as** ADC resolution **and** analog trigger **are defined in the Glossary.***

MIO-16/16D Data Acquisition

The MIO-16/16D can perform single-channel data acquisition and multiple-channel scanned data acquisition. For single-channel data acquisition, you select a single analog input channel and gain setting. The MIO-16/16D performs a single A/D conversion on that channel every sample interval.

For multiple-channel scanned data acquisition, the MIO-16/16D scans a set of analog input channels, each with its own gain setting. In this method, a scan sequence indicates which analog channels to scan and the gain settings for each channel. The length of this scan sequence can be 1 to 16 channel/gain pairs. During scanning, the analog input circuitry performs an A/D conversion on the next entry in the scan sequence. The MIO-16/16D performs an A/D conversion once every sample interval. For maximum performance, this operation is pipe line

so that the device switches to the next channel while the current A/D conversion is performed. When the end of the scan sequence is reached, the MIO-16/16D waits for a specified scan interval before scanning the channels again. The device scans the channels repeatedly at the beginning of each scan interval until the device acquires the required number of samples. For example, you can scan a sequence of four channels once every 10 s. The MIO-16/16D could sample the channels at the beginning of the 10 s interval, within 100 µs, with a 25 µs sample interval between channels. If you set the scan interval to 0, the scan sequence starts over again immediately at the end of each scan sequence without waiting for a scan interval. The 0 scan interval setting causes the MIO-16/16D to scan the channels repeatedly as fast as possible.

You can combine both single-channel and multiple-channel acquisition with any of the following additional modes:

- Posttrigger mode
- Pretrigger mode
- Double-buffered mode
- AMUX-64T mode
- SCXI mode

Posttrigger mode collects a specified number of samples after the MIO-16/16D receives a trigger. You can initiate posttrigger acquisition through software or by applying a pulse edge to the STARTTRIG* input. After the user-specified buffer is full, the data acquisition stops.

Pretrigger mode collects data both before and after the MIO-16/16D receives a trigger. You can initiate data acquisition as in posttrigger mode, either through software or by applying a pulse on STARTTRIG*. The device collects samples and fills the user-specified buffer without stopping or counting samples until the device receives a pulse at the STOPTRIG input. The device then collects a specified number of samples and stops the acquisition. The buffer is treated as a circular buffer—when the entire buffer has been written to, data is stored at the beginning again, overwriting the old data. When data acquisition stops, the buffer has samples from before and after the stop trigger occurred. The number of samples saved depends on the length of the user-specified buffer and on the number of samples specified to be acquired after receipt of the trigger.

Double-buffered mode, like pretrigger mode, also fills the user-specified buffer continuously. You can call `DAQ_DB_Transfer` or `DAQ_Monitor` to transfer old data into a second buffer before it is overwritten by new data. `DAQ_DB_Transfer` transfers data out of one half of the buffer while the other half is filled with new data.

In the AMUX-64T mode, you use one or more external AMUX-64T devices to extend the number of analog input channels available. You connect the external signals to the pins of the AMUX-64T devices instead of directly to the pins of the DAQ device.

You can use SCXI modules as a data acquisition front end for the MIO-16/16D to condition the input signals and multiplex the channels. You can use all the modes just described in conjunction with SCXI. *The SCXI Hardware* section later in this chapter describes how to use the SCXI functions to set up the SCXI modules for a data acquisition application.

☞ **Note:** ***Refer to the*** `Set_DAQ_Device_Info` ***function in Chapter 2,*** *Function Reference****, of the*** *NI-DAQ Function Reference Manual for PC Compatibles* ***for information on data transfer modes.***

MIO-16/16D Data Acquisition Timing

The onboard Am9513 Counter/Timer can perform timing for data acquisition, or you can perform timing externally. The Am9513 16-bit Counter/Timers are assigned as follows:

- Counter 1 is used for multiple-channel scanning with the AMUX-64T and is otherwise available for general-purpose counting functions.
- Counter 2 is used for multiple-channel scanning when **scan_interval** is not equal to 0. Counter 2 is also used for waveform generation and later update mode analog output. Counter 2 is otherwise available for general-purpose counting functions.
- Counter 3 is a sample-interval counter reserved for data acquisition.
- Counter 4 is a sample counter reserved for data acquisition.
- Counter 5 is available for general-purpose counting functions.

Data acquisition timing involves the following timing signals:

- A *start trigger* is an edge-triggered signal that initiates a data acquisition sequence. You can supply a trigger pulse either externally through the I/O connector STARTTRIG* input or from software control. You can enable a hardware start trigger by calling `DAQ_Config`.
- A *conversion pulse* is a signal that generates a pulse once every sample interval, initiating an A/D conversion. The onboard, programmable sample-interval clock supplied by the Am9513 Counter/Timer on the MIO-16/16D and AT-MIO-16D can generate this signal, or you can supply it externally through the I/O connector EXTCONV* input. You can select external conversion pulses by calling `DAQ_Config`. If you do not want to use external conversion pulses, you should disconnect the EXTCONV* pin on the I/O connector to prevent extra conversions. If you are using SCXI with your DAQ device, refer to the *SCXI Data Acquisition Rates* section in Chapter 3, *Software Overview*, for the effect of SCXI module settling time on your DAQ device rates.
- You use a *sample counter* when conversion pulses are generated either by the onboard sample-interval counter or externally. The sample counter tallies the number of A/D conversions (samples) and shuts down the data acquisition timing circuitry when the number of samples you want has been acquired.
- A *stop trigger* is a signal you use for pretriggered data acquisition to notify the MIO-16/16D to stop acquiring data after a specified number of samples. Until you apply the stop trigger pulse at the STOPTRIG input on the I/O connector, a data acquisition operation remains in a continuous acquisition mode, indefinitely writing and rewriting data to the buffer. You can select pretriggering by calling `DAQ_StopTrigger_Config`.
- A *timebase clock* is a clock signal that is the timebase for the sample-interval counter. Onboard selections of 1 MHz, 100 kHz, 10 kHz, 1 kHz, or 100 Hz are available. The I/O connector at the SOURCE5 input can also supply an external timebase clock.

See your device user manual for more information regarding these signals.

MIO-16/16D Data Acquisition Rates

Refer to the appropriate device user manual for the data acquisition rates and sample intervals for single-channel data acquisition with the MIO-16/16D.

If you are using SCXI with your DAQ device, refer to the *SCXI Data Acquisition Rates* section in Chapter 3, *Software Overview*, for the effect of SCXI module settling time on your DAQ device rates.

With multiple-channel scanned data acquisition, extra time is required by the data acquisition circuitry for gain/multiplexer settling because of channel switching. The settling time required depends on the gain setting used for each channel. This settling time also limits data acquisition rates. Refer to the appropriate device user manual for the recommended values for settling time versus gain.

E Series, AT-MIO-16F-5, AT-MIO-64F-5, and AT-MIO-16X Data Acquisition

The E Series, AT-MIO-16F-5, AT-MIO-64F-5, and AT-MIO-16X can perform single-channel data acquisitions and multiple-channel scanned data acquisitions. For single-channel data acquisition, select a single analog input channel and gain setting. The device performs a single A/D conversion on that channel every sample interval.

For multiple-channel scanned data acquisition, the device scans a set of analog input channels, each with its own gain setting. In this method, a scan sequence indicates which analog channels to scan and the gain settings for each channel. The length of this scan sequence can be 1 to 512 channel/gain pairs. During scanning, the analog input circuitry performs an A/D conversion on the next entry in the scan sequence. The device performs an A/D conversion once every sample interval. For maximum performance, this operation is pipelined so that the device switches to the next channel while the current A/D conversion is performed. The device waits for a specified scan interval before scanning the channels again. The channels are scanned repeatedly at the beginning of each scan interval until the required number of samples has been acquired. For example, you can scan a sequence of four channels once every 10 s. The device could sample the channels at the beginning of the 10 s interval, within 20 µs, with a 5 µs sample interval between channels. If you set the scan interval to 0, the scan sequence starts over again immediately at the end of each scan sequence without waiting for a scan interval. The 0 scan interval

setting causes the device to scan the channels repeatedly as fast as possible.

You can combine both single-channel and multiple-channel acquisition with any of the following additional modes:

- Posttrigger mode
- Pretrigger mode
- Double-buffered mode
- AMUX-64T mode
- SCXI mode

Posttrigger mode collects a specified number of samples after the device receives a trigger. Refer to the *start trigger* discussion in the appropriate data acquisition timing section for your device later in this chapter for details. After the user-specified buffer is full, the data acquisition stops.

Pretrigger mode collects data both before and after the device receives a trigger in posttrigger mode, either through software or by applying a hardware signal. The device collects samples and fills the user-specified buffer without stopping until the device receives the *stop trigger* signal. Refer to the *stop trigger* discussion in the appropriate data acquisition timing section for your device later in this chapter for details. The device then collects a specified number of samples and stops the acquisition. The buffer is treated as a circular buffer—when the entire buffer has been written to, data is stored at the beginning again, overwriting the old data. When data acquisition stops, the buffer has samples from before and after the stop trigger occurred. The number of samples saved depends on the length of the user-specified buffer and on the number of samples specified to be acquired after receipt of the trigger.

Double-buffered mode, like pretrigger mode, also fills the user-specified buffer continuously. You can call `DAQ_DB_Transfer` to transfer old data into a second buffer before it is overwritten by new data. `DAQ_DB_Transfer` transfers data out of one half of the buffer while the other half is filled with new data.

In the AMUX-64T mode, you use one or more external AMUX-64T devices to extend the number of analog input channels available. You connect the external signals to the pins of the AMUX-64T devices, instead of directly to the pins of the DAQ device.

You can use SCXI modules as a data acquisition front end for the device to condition the input signals and multiplex the channels. You can use all the modes just described in conjunction with SCXI. The *SCXI Modules and Compatible DAQ Devices* section later in this chapter describes how to use the SCXI functions to set up the SCXI modules for a data acquisition to be performed by a DAQ device.

☞ **Note:** ***Refer to the*** `Set_DAQ_Device_Info` ***function in Chapter 2,*** *Function Reference****, of the*** *NI-DAQ Function Reference Manual for PC Compatibles* ***for information on data acquisition modes.***

☞ **Note:** ***(AT-MIO-64F-5 and AT-MIO-16X only) If you have an AT-DSP2200, you can acquire AT-MIO-64F-5 or AT-MIO-16X data directly into DSP memory through the RTSI bus. Allocate a DSP buffer through*** `NI_DAQ_Mem_Alloc` ***and pass the buffer to NI-DAQ through*** `DAQ_Start` ***or*** `SCAN_Start`**.**

E Series Data Acquisition Timing

The following DAQ-STC counters are used for data acquisition timing and control:

The *scan counter* is used to control the number of scans you will acquire. If you want to perform pretriggered acquisition, this counter will ensure that you acquire selected number of scans before the stop trigger is recognized.

- The *scan timer* is a counter that you can use for *start scan* timing.
- The *sample interval timer* is a counter that you can use for *conversion* timing.

Data acquisition timing involves the following timing signals:

- A *start trigger* is a signal that initiates a data acquisition sequence. You can supply this signal externally through a selected I/O connector pin, through a RTSI bus trigger line, or by software.
- A *start scan* signal initiates individual scans. This signal can be supplied from the on-board programmable scan timer, externally through a selected I/O connector pin, through a RTSI bus trigger line, or by software.
- A *conversion* signal initiates individual analog-to-digital (A/D) conversions. This signal can be supplied from the on-board programmable sample timer, externally through a selected I/O connector pin, through a RTSI bus trigger line, or by software.

- A *stop trigger* is a signal used for pretriggered data acquisition to notify your device to stop acquiring data after a specified number of scans. Data acquisition operation is continuously performed until the device receives this signal. This signal can be supplied externally through a selected I/O connector pin, through a RTSI bus trigger line, or by software.
- *Gate* is a signal used for gating the data acquisition. When you enable gating, the data acquisition will proceed only on selected level of the gate signal. This signal can be supplied externally through a selected I/O connector pin, through a RTSI bus trigger line.
- *Scan timer timebase* is a signal used by the scan timer for scan interval timing. This signal is used only when the scan timer is used. This signal can be supplied from one of the on-board timebase sources, externally through a selected I/O connector pin, or through a RTSI bus trigger line.
- *Sample interval timer timebase* is a signal used by the sample interval timer for conversion timing. This signal is used only when the sample interval timer is used. This signal can be supplied from one of the on-board timebase sources, externally through a selected I/O connector pin, or through a RTSI bus trigger line.

See your DAQ device user manual for more information regarding these signals.

DAQ devices with the DAQ-STC use two counters, the scan interval counter and the sample interval counter. The E Series devices support both internal and external timebases. The internal timebases available on the DAQ-STC are 20 MHz (50 ns) and 100 kHz (10 µs). The scan interval counter is a 24-bit counter, and the sample interval counter is a 16-bit counter.

While the scan interval counter has the freedom to work with both internal and external timebases, the sample interval counter can use either the 20 MHz timebase or the timebase used by the scan interval counter.

Whenever you specify a timebase value different from the internal timebases the DAQ-STC uses, NI-DAQ attempts to convert the timebase and interval values you specified into those that the DAQ-STC can use. If NI-DAQ cannot make the translation without a loss of resolution, it returns **rateNotSupportedErr**. This would typically occur if you specified a timebase of 5 (100 Hz) and a sample

interval of 100, for example, for a resulting sample interval of 1 s. This would generate an error because the sample interval counter would roll over before 1 s.

AT-MIO-16F-5, AT-MIO-64F-5, and AT-MIO-16X Data Acquisition Timing

Timing for data acquisition can be performed by the onboard Am9513 Counter/Timer or externally. The Am9513 16-bit Counter/Timers are assigned as follows:

- Counter 1 is used for multiple-channel scanning with the AMUX-64T, and for waveform generation. Counter 1 is otherwise available for general-purpose counting functions.
- Counter 2 is used for multiple-channel scanning when **scan_interval** is not equal to 0, and for waveform generation. Counter 2 is otherwise available for general-purpose counting functions.
- Counter 3 is a sample-interval counter reserved for data acquisition.
- Counter 4 is a sample counter reserved for data acquisition.
- Counter 5 is available for general-purpose counting functions, and is also used for waveform generation.

Although counters 1, 2, and 5 are all mentioned as being used for waveform generation on the AT-MIO-16F-5, AT-MIO-64F-5, and AT-MIO-16X, only one counter at a time is used. The choice is based on availability with counter 5 being the first choice, counter 2 second, and counter 1 last.

Data acquisition timing involves the following timing signals:

- A *start trigger* is an edge-triggered signal that initiates a data acquisition sequence. A trigger pulse can be supplied either externally through the I/O connector EXTTRIG* input or from software control. You can enable a hardware start trigger by calling `DAQ_Config`.
- A *conversion pulse* is a signal that generates a pulse once every sample interval, causing an A/D conversion to be initiated. This signal can be generated by the onboard, programmable sample-interval clock supplied by the Am9513 Counter/Timer on board, or can be supplied externally through the I/O connector EXTCONV* input. You can select external conversion pulses by calling

DAQ_Config. If you do not want to use external conversion pulses, you should disconnect the EXTCONV* pin on the I/O connector to prevent extra conversions.

- A *sample counter* is used when conversion pulses are generated either by the onboard sample-interval counter or externally. The sample counter tallies the number of A/D conversions (samples) and shuts down the data acquisition timing circuitry when the device has acquired the desired number of samples.
- A *stop trigger* is a signal used for pretriggered data acquisition to notify the device to stop acquiring data after a specified number of samples. Until the stop trigger pulse is applied at the EXTTRIG input on device, a data acquisition operation remains in a continuous acquisition mode, indefinitely writing and rewriting data to the buffer. You can select pretriggering by calling DAQ_StopTrigger_Config.
- A *timebase clock* is a clock signal that provides the timebase for the sample-interval counter. Onboard selections of 5 MHz, 1 MHz, 100 kHz, 10 kHz, 1 kHz, or 100 Hz are available. You can also apply an external timebase clock through the I/O connector at the SOURCE5 input.

See your DAQ device user manual for more information regarding these signals.

E Series, AT-MIO-16F-5, AT-MIO-64F-5, and AT-MIO-16X DAQ Rates

Refer to the appropriate user manual for single-channel and multiple-channel DAQ rates and settling accuracy.

If you are using SCXI with your DAQ device, refer to the *SCXI Data Acquisition Rates* section in Chapter 3, *Software Overview*, for the effect of SCXI module settling time on your DAQ device rates.

MIO Device Analog Output

The MIO devices contain two analog output channels numbered 0 and 1. Each analog output channel contains a 12-bit DAC, except on the AT-MIO-16X, which contains a 16-bit DAC. You can configure each analog output channel for unipolar or bipolar voltage output except on the MIO-16XE-50 devices, which is always in bipolar mode. An onboard voltage reference of +10 V is available for the analog output channels. An external reference voltage signal can also drive the

analog output channels except on the MIO-16XE-50 devices. See the `AO_Configure` function in Chapter 2, *Function Reference*, of the *NI-DAQ Function Reference Manual for PC Compatibles* for more information about configuring the DACs.

MIO Device Waveform Generation

The Waveform Generation functions can continuously write values to either one or both analog output channels using an onboard or external clock to update the DACs at regular intervals. The values are contained in a buffer that you allocate and fill. The resultant voltages produced at the analog output channels depend on the value of the numbers in the buffer, the level of the reference voltage, and the polarity setting.

☞ **Note:** ***(MIO E Series, AT-MIO-16F-5, AT-MIO-64F-5, and AT-MIO-16X only) NI-DAQ can use either DMA or interrupt service routines to generate waveforms on the analog output channels. By default, NI-DAQ uses DMA because DMA is simply more efficient. If you prefer to reserve the DMA channels for the Data Acquisition functions or for other devices in the system, call*** `Set_DAQ_Device_Info` ***(described in Chapter 2,*** *Function Reference****, of the*** *NI-DAQ Function Reference Manual for PC Compatibles****) to force NI-DAQ not to use DMA for waveform generation.***

MIO E Series Waveform Generation Using Onboard Memory

The MIO E Series devices with analog output FIFOs (the AT-MIO-16E-1, AT-MIO-16E-2, NEC-MIO-16E-4, and AT-MIO-64E-3) support FIFO mode waveform generation. In this mode, waveform data is transferred to the onboard DAC FIFO memory only once. These values are then cycled through to generate the waveform continuously or for a finite number of iterations. No interrupt service or DMA operation is required to transfer more data to the device during waveform generation.

The following conditions must be satisfied to use FIFO mode waveform generation:

- One cycle worth of waveform data can be held entirely on the device DAC FIFO memory.
- Double-buffered waveform generation mode is disabled.
- The number of cycles to generate can be infinite (denoted by the value 0), or between 1 and 16,777,216 inclusive.

There are two variations possible in FIFO mode waveform generation:

- Continuous cyclic waveform generation—An onboard counter or an external signal provides the update pulses. Once started, the waveform generation continues until you call the clear function to stop the waveform.
- Programmed cyclic waveform generation—An onboard counter or an external signal provides the update pulses. A separate onboard counter terminates the waveform after a finite number of cycles has been generated.

Refer to your DAQ device manual for more information on waveform generation.

AT-MIO-16X and AT-MIO-64F-5 Waveform Generation Using Onboard Memory

The AT-MIO-16X and AT-MIO-64F-5 support FIFO mode waveform generation. In this mode, waveform data is transferred to onboard DAC FIFO memory only once. These values are then cycled through to generate the waveform continuously or for a finite number of iterations. No interrupt service or DMA operation is required to transfer more data to the device during waveform generation.

The following conditions must be satisfied to use FIFO mode waveform generation:

- One cycle worth of waveform data can be held entirely on the device DAC FIFO memory.
- Double-buffered waveform generation mode is disabled.
- The number of cycles to generate can be infinite (denoted by the value 0), or between 1 and 65,535 inclusive.

There are three variations possible in FIFO mode waveform generation:

- Continuous cyclic waveform generation—An onboard counter (counter 1, 2, 3, or 5 of the Am9513 Counter/Timer) or an external signal provides the update pulses. Once started, the waveform generation continues until you call the clear function to stop the waveform.
- Programmed cyclic waveform generation—An onboard counter (counter 1, 2, 3, or 5 of the Am9513 Counter/Timer) or an external signal provides the update pulses. A separate onboard counter

(counter 1, 2, or 5) terminates the waveform after a finite number of cycles has been generated.

- Pulsed waveform generation—An available onboard counter or an external signal provides the update pulses. Counter 1 of the Am9513 Counter/Timer counts the number of iterations to generate before the delay phase. At the delay phase, counter 2 or an external signal provides a delay period before the waveform is restarted for the same number of iterations and delay again. Once started, this sequence continues until you call the clear function to stop the waveform.

Refer to your DAQ device manual for more information on waveform generation.

Am9513-Based Device Digital I/O

The MIO devices contain eight bits of digital I/O. These bits are divided into a set of two digital I/O ports of four bits each. The 4-bit digital I/O ports are labeled as ports DIOA and DIOB. These ports are referred to as ports 0 and 1 by the Digital I/O functions, in which:

- port DIOA = port 0
- port DIOB = port 1

You can configure port 0 or 1 as either an input port or an output port. Any port that you configure as an output port has read-back capability (that is, by reading the port, you can determine what digital value the output port is currently asserting). The MIO device digital I/O ports operate in nonlatched mode only.

☞ **Note:** ***Connecting one or more AMUX-64T devices or an SCXI chassis to an MIO device renders port DIOA unavailable.***

NI-DAQ also reserves line 0 of port DIOB for input from the SCXI hardware if you have SCXI configured. The remaining lines of port DIOB are available for input only.

In addition to the eight bits of digital I/O described above, the AT-MIO-16D contains another 24 bits of digital I/O. The additional 24 bits of digital I/O are the same as those of the DIO-24 device described later in this chapter.

For a description of the AT-MIO-16D digital I/O, see *The DIO-24, AT-MIO-16D, and AT-MIO-16DE-10 Digital I/O Devices* section later in this chapter.

E Series Digital I/O

The E Series devices contain one 8-bit digital I/O port supplied by the DAQ-STC chip. This port is referred to as port 0 by the Digital I/O functions.

You can configure the entire digital port as either an input or an output port, or you can configure individual lines for either input or output. The port has read-back capability (that is, by reading the port, you can determine what digital value the output port is currently asserting). This port operates in nonlatched mode only.

☞ **Note:** ***Connecting one or more AMUX-64T devices or an SCXI chassis to an E Series device renders various lines of the digital I/O port unavailable:***

One AMUX-64T device—Lines 0 and 1 are unavailable.
Two AMUX-64T devices—Lines 0, 1, and 2 are unavailable.
Four AMUX-64T devices—Lines 0, 1, 2, and 3 are unavailable.
SCXI—Lines 0, 1, 2, and 4 are unavailable.

The remaining lines of the digital I/O port are available for input or output. You should use `DIG_Line_Config` ***to configure these remaining lines.***

The AT-MIO-16DE-10 has one 8-bit line-configurable port named port 0 that does not support handshaking or asynchronous operations. The DAQ-STC chip supplies this port. The AT-MIO-16DE-10 also has three 8-bit ports named ports 2, 3, and 4 that do support handshaking and asynchronous operations and are directionally configurable only on a per-port basis. An 8255 chip supplies these three ports. The three 8255 ports are numbered 2, 3, and 4 for compatibility with the same three ports on the AT-MIO-16D, an older product. The following table summarizes the four ports available on the AT-MIO-16DE-10:

Port	Size	Configuration	Type	Supplied by
0	8-bit	Line configurable	Immediate	DAQ-STC chip
2	8-bit	Handshaking	Asynchronous or immediate	8255 chip

Port	Size	Configuration	Type	Supplied by
3	8-bit	Handshaking	Asynchronous or immediate	8255 chip
4	8-bit	n/a	Immediate	8255 chip

For a description of the AT-MIO-16DE-10 digital I/O, see *The DIO-24, AT-MIO-16D, and AT-MIO-16DE-10 Digital I/O Devices* section later in this chapter.

Am9513-Based Device Counter/Timer Operation

The MIO devices contain an onboard Am9513 System Timing integrated circuit that has five independent 16-bit counter/timers and a 4-bit programmable frequency output. Figure 2-1 diagrams the 16-bit counters available on the MIO devices.

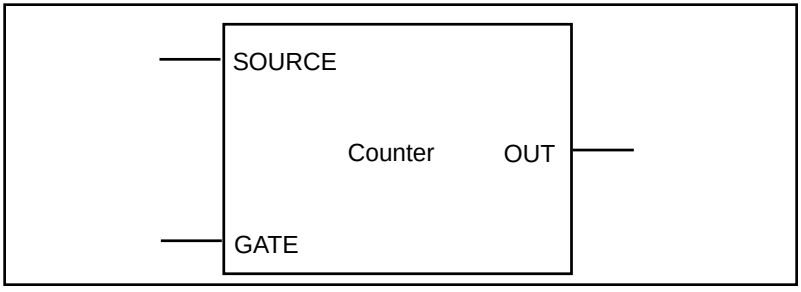


Figure 2-1. Counter Block Diagram

Each counter has a SOURCE input, a GATE input, and an output labeled OUT.

The counters can use several timebases for counting operations. A counter can use the signal supplied at any of the Am9513 five SOURCE or GATE inputs for counting operations. The Am9513 also makes available five internal timebases that any counter can use:

- 1 MHz clock (1 µs resolution)
- 100 kHz clock (10 µs resolution)
- 10 kHz clock (100 µs resolution)
- 1 kHz clock (1 ms resolution)
- 100 Hz clock (10 ms resolution)

In addition, you can program the counter to use the output of the next lower-order counter as a signal source. This arrangement is useful for counter concatenation. For example, you can program counter 2 to count the output of counter 1, thus creating a 32-bit counter.

You can configure a counter to count either falling or rising edges of the selected internal timebase, SOURCE input, GATE input, or the next lower-order counter signal.

You can use the counter GATE input to gate counting operations. After you configure a counter through software for an operation, you can use a signal at the GATE input to start and stop the counter operation. There are eight gating modes available in the Am9513:

- No Gating—the counter is started and stopped by software.
- High-Level Gating—the counter is active when its gate input is at high-logic state. The counter is suspended when its gate input is at low-logic state.
- Low-Level Gating—the counter is active when its gate input is at low-logic state. The counter is suspended when its gate input is at high-logic state.
- Rising Edge Gating—the counter starts counting when it receives a low-to-high edge at its gate input.
- Falling Edge Gating—the counter starts counting when it receives a high-to-low edge at its gate input.
- High Terminal Count Gating—the counter is active when the next lower-order counter reaches terminal count (TC) and generates a TC pulse.
- High-Level Gate N+1 Gating—the counter is active when the gate input of the next higher-order counter is at high-logic state. Otherwise, the counter is suspended.
- High-Level Gate N-1 Gating—the counter is active when the gate input of the next lower-order counter is at high-logic state. Otherwise, the counter is suspended.

Counter operation starts and stops relative to the selected timebase. When you configure a counter for no gating, the counter starts at the first timebase/source edge (rising or falling, depending on the selection) after the software configures the counter. When you configure a counter for gating modes, gate signals take effect at the next timebase/source edge. For example, if you configure a counter to count rising edges and to use the falling edge gating mode, the counter

starts counting on the next rising edge after it receives a high-to-low edge on its GATE input. Thus, some time is spent synchronizing the GATE input with the timebase/source. This synchronization time creates a time lapse uncertainty from zero to one timebase period between the application of the signal at the GATE input and the start of the counter operation.

The counter generates timing signals at its OUT output. If the counter is not operating, you can set its output to one of three states—high-impedance state, low-logic state, or high-logic state.

The counters generate two types of output signals during counter operation: TC pulse output and TC toggled output. A counter reaches TC when it counts up (to 65,535) or down (to 0) and rolls over. In many counter applications, the counter reloads from an internal register when it reaches TC. In TC pulse output mode, the counter generates a pulse during the cycle in which it reaches TC. In TC toggled output mode, the counter output changes state on the next source edge after reaching TC. In addition, you can configure the counters for positive logic output or negative (inverted) logic output. Figure 2-2 shows examples of the four types of output signals generated.

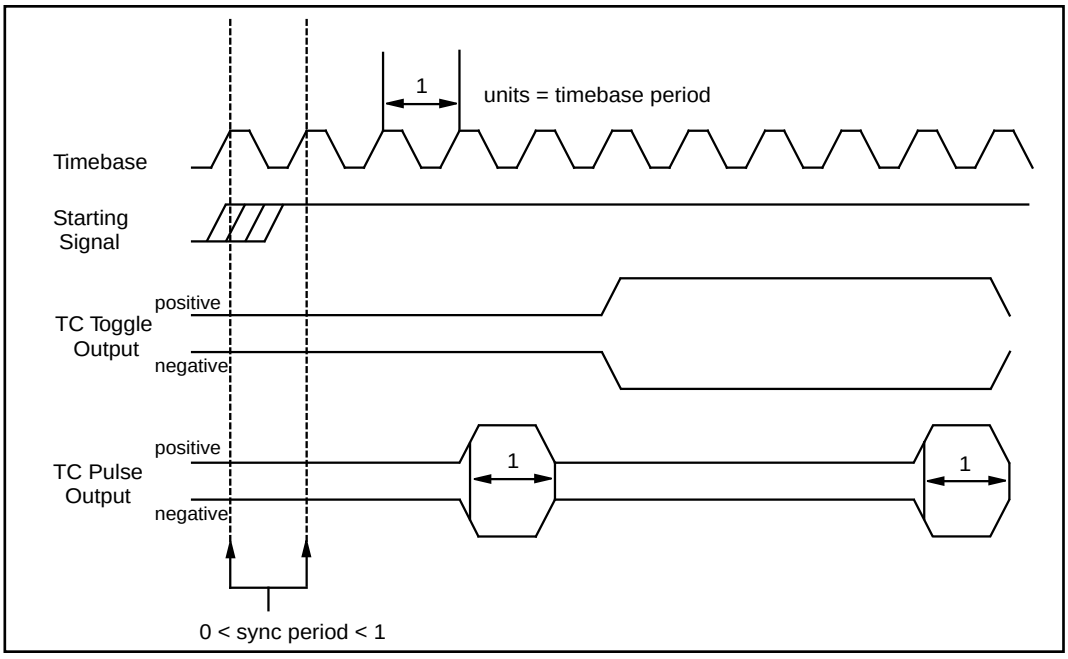


Figure 2-2. MIO Device Counter Timing and Output Types

Figure 2-2 represents a counter generating a delayed pulse and demonstrates the four forms the output pulse can take given the four different types of output signals supported. The TC toggled positive logic output looks like what would be expected when generating a pulse. For most of the Counter/Timer functions, TC toggled output is the preferred output configuration; however, the other signal types are also available. The starting signal shown in Figure 2-2 represents either a software starting of the counter, for the No-Gating mode, or some sort of signal at the GATE input. The signal could be either a rising edge gate or a high-level gate. If the signal is a low-level or falling edge gate, the starting signal simply appears inverted. In Figure 2-2, the counter is configured to count the rising edges of the timebase; therefore, the starting signal takes effect on the rising edge of the timebase, and the signal output changes state with respect to the rising edge of the timebase.

Programmable Frequency Output Operation

The Am9513-based devices have a 4-bit programmable frequency output signal. This signal is a divided-down version of the selected timebase. Any of five internal timebases, counter SOURCE inputs, and counter GATE inputs can be selected as the FOUT source. See the `CTR_FOUT_Config` function in Chapter 2, *Function Reference*, of the *NI-DAQ Function Reference Manual for PC Compatibles* for FOUT use and timing.

Am9513-Based Device Counter/Timer Usage

☞ **Note:** ***This section does not apply to the E Series devices.***

NI-DAQ uses the five counter/timers as follows:

- Counter 1 is used for multiple-channel scanning with the AMUX-64T or an SCXI chassis, and for waveform generation on the AT-MIO-16F-5, AT-MIO-64F-5, and AT-MIO-16X. Counter 1 is otherwise available for general-purpose counting functions.
- Counter 2 is used for multiple-channel interval scanning, and waveform generation and later update mode. You can reserve counter 2 for Track*/Hold manipulation of the SCXI-1140. Counter 2 is otherwise available for general-purpose counting functions.
- Counter 3 is a sample-interval counter always reserved for data acquisition.
- Counter 4 is a sample counter always reserved for data acquisition.
- Counter 5 is used for the data acquisition sample counter when the number of samples exceeds 65,535, and for waveform generation on AT-MIO-16F-5, AT-MIO-64F-5, and AT-MIO-16X. Counter 5 is otherwise available for general-purpose counting functions.

Although counters 1, 2, and 5 are all mentioned as being used for waveform generation on the AT-MIO-16F-5, AT-MIO-64F-5, and AT-MIO-16X, only one of these counters is used at a time. The choice is based on availability, with counter 5 being the first choice, counter 2 second, and counter 1 last.

Some of the three available counter signals are connected to the I/O connector and to the RTSI bus. Figure 2-3 shows the connections on an

AT-MIO-16 or AT-MIO-16D, and Figure 2-4 shows the connections on an AT-MIO-16F-5, AT-MIO-64F-5, or AT-MIO-16X.

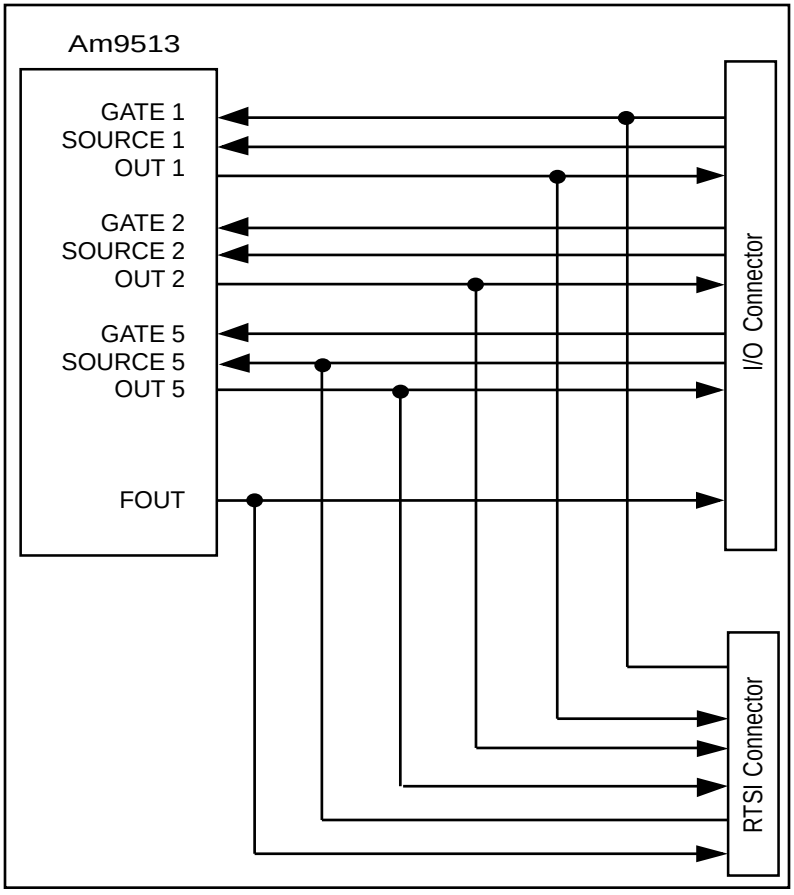


Figure 2-3. AT-MIO-16 and AT-MIO-16D Counter/Timer Signal Connections

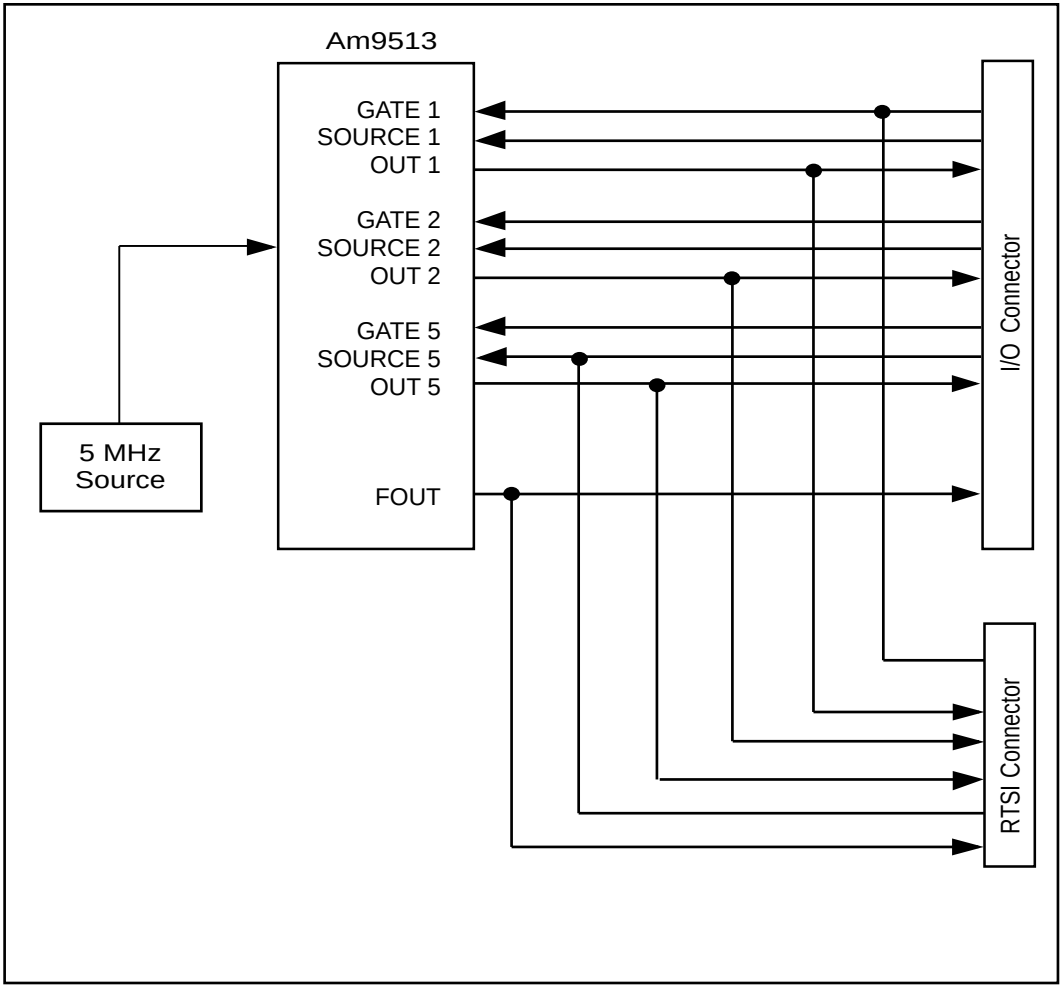


Figure 2-4. AT-MIO-16F-5, AT-MIO-64F-5, and AT-MIO-16X Counter/Timer Signal Connections

E Series Counter/Timer Operation

The E Series devices use the National Instruments DAQ-STC counter/timer chip. The DAQ-STC has two 24-bit counter/timers that are always available for general-purpose counter/timer applications. Refer to the GPCTR functions in Chapter 2, *Function Reference*, of the *NI-DAQ Function Reference Manual for PC Compatibles* for more information.

E Series Programmable Frequency Output Operation

The E Series devices have one I/O connector pin (FREQ_OUT) capable of outputting a programmable frequency signal. This signal is a divided-down version of the selected timebase. Available timebases are 10 MHz and 100 kHz, and you can divide them by numbers 1 through 16.

E Series PFI Pins

The PFI pins on the E Series devices allow you to programmatically route internal device signals to and from the I/O connector. Refer to your E Series device user manual for a complete discussion of PFI pins.

The Lab-PC+, SCXI-1200, DAQPad-1200, and DAQCard-1200 Multifunction I/O Devices

☞ **Note:** ***The Lab-PC+, SCXI-1200, DAQPad-1200, and DAQCard-1200 are very similar in function and are referred to ONLY in this section as Lab boards. Where specific differences occur, the product names will be used.***

Lab Board Analog Input

The Lab boards contain eight single-ended analog input channels numbered 0 through 7. The analog input channels are multiplexed into a single programmable gain stage and 12-bit ADC. The Lab boards have gains of 1, 2, 5, 10, 20, 50, and 100.

You can configure the input channels to be differential or nonreferenced single ended. In differential mode, four channels are available, namely, 0, 2, 4, and 6. Configure the mode via jumper W4 on the Lab-PC+. The SCXI-1200, DAQPad-1200, and DAQCard-1200 support software configuration of the input mode.

You can hardware jumper-configure analog input on the Lab-PC+ and software configure analog input on the SCXI-1200, DAQPad-1200, and DAQCard-1200 for two different input ranges:

- 0 to +10 V (unipolar)
- -5 to +5 V (bipolar)

You can initiate A/D conversions through software or by applying active low pulses to the EXTCONV* input on the Lab board I/O connector. Each of the Lab boards has a FIFO on the device in which to temporarily store the results of A/D conversions. The FIFO size differs as follows:

Device	FIFO Size (in words)
Lab-PC+	256
SCXI-1200	2,048
DAQPad-1200	2,048
DAQCard-1200	2,048

Lab Board Data Acquisition

The Lab boards can perform single-channel data acquisition and multiple-channel scanned data acquisition. For single-channel data acquisition, you select a single analog input channel and gain setting. The board performs a single A/D conversion on that channel every sample interval.

For multiple-channel scanned data acquisition, the Lab boards scan a sequence of analog input channels. A single gain setting is used for all channels scanned. A *sample interval* indicates the time to elapse between A/D conversions on each channel in the sequence. You need only to select a single starting channel to select the sequence of channels to scan. The board then scans the channels in consecutive order until channel 0 is reached, and then the scan begins anew with the starting channel. For example, if the starting channel is channel 3, the scan sequence is as follows:

channel 3, channel 2, channel 1, channel 0, channel 3, and so on

The Lab boards support interval scanning. A scan interval is the time that elapses between two channel-scanning cycles.

You can combine both the single-channel and multiple-channel acquisitions with any of the following additional modes:

- Posttrigger mode
- Pretrigger mode

- Double-buffered mode
- SCXI mode (Lab-PC+, SCXI-1200, and DAQCard-1200 only)

Posttrigger mode collects a specified number of samples after the board receives a trigger. You can initiate posttrigger acquisition through software or when a pulse edge is applied to the Lab board EXTTRIG input. After the user-specified buffer has been filled, the data acquisition stops.

Pretrigger mode collects data both before and after the board receives a trigger. You can initiate data acquisition through software. The Lab board collects samples and fills the user-specified buffer without stopping or counting samples until the board receives a pulse at the EXTTRIG input. The Lab board then collects a specified number of samples and stops the acquisition. The buffer is treated as a circular buffer—when the entire buffer has been written to, data is stored at the beginning again, overwriting the old data. When data acquisition stops, the buffer has samples from before and after the stop trigger occurred. The number of samples saved depends on the length of the user-specified buffer and on the number of samples specified to be acquired after receipt of the trigger.

Because there is only one EXTTRIG input on the Lab board I/O connector, a single acquisition cannot employ both of these trigger modes.

Double-buffered mode, like pretrigger mode, also fills the user-specified buffer continuously. Unlike pretrigger mode, however, double-buffered mode transfers old data into a second buffer before overwriting the old data with new data. Data is transferred out of one half of the buffer while the other half is being filled with new data. You can use double-buffered mode in conjunction with either pretrigger or posttrigger modes.

You can use SCXI modules as a data acquisition front end for the Lab-PC+, SCXI-1200, and DAQCard-1200 to signal condition the input signals and multiplex the channels. You can use all the modes just described in conjunction with SCXI for single-channel acquisitions; however, multiple-channel acquisitions are only supported when using the SCXI-1120 or SCXI-1121 modules in Parallel mode. *The SCXI Hardware* section later in this chapter describes how the SCXI functions are used to set up the SCXI modules for a data acquisition to be performed by a DAQ device.

☞ **Note:** ***Refer to the*** `Set_DAQ_Device_Info` ***function in Chapter 2,*** *Function Reference****, of the*** *NI-DAQ Function Reference Manual for PC Compatibles****, for information on data transfer modes.***

Lab Board Data Acquisition Timing

Timing for data acquisition can be performed by the two onboard 8253 Counters/Timers or externally. Each 8253 Counter/Timer has three independent 16-bit counters/timers.

- Counter A0 is a sample-interval counter reserved for data acquisition.
- Counter A1 is a sample counter reserved for data acquisition.
- Counter A2 is an update-interval counter reserved for waveform generation.
- Counter B0 is used for extending the timebase for data acquisition or waveform generation when the interval between samples or updates is greater than 65,535.
- Counter B1 is available for general-purpose counting functions. However, it is reserved during data acquisition interval scanning.
- Counter B2 is available for general-purpose counting functions.

Data acquisition timing involves the following timing signals:

- A *start trigger* is an edge-triggered signal that initiates a data acquisition sequence. You can supply a trigger pulse either externally through the I/O connector EXTTRIG input or from software control. You can enable a hardware start trigger by calling `DAQ_Config`.
- A *conversion pulse* is a signal that generates a pulse once every sample interval, causing an A/D conversion to be initiated. This signal can be generated by the onboard, programmable sample-interval clock supplied by the 8253 Counter/Timer on the Lab board, or can be supplied externally through the I/O connector EXTCONV* input. You can select external conversion pulses by calling `DAQ_Config`. If you do not want to use external conversion pulses, you should disconnect the EXTCONV* pin on the I/O connector to prevent extra conversions.
- A *scan-interval pulse* is a signal that generates a pulse every scan. The scan clock for interval scanning pulses can be generated by the onboard, programmable scan-interval counter, or can be supplied through the OUTB1 pin on the I/O connector.

- A *sample counter* is used when conversion pulses are generated either by the onboard sample-interval counter or externally. The sample counter tallies the number of A/D conversions (samples) and shuts down the data acquisition timing circuitry when the board has acquired the desired number of samples.
- A *stop trigger* is a signal used for pretriggered data acquisition to notify the Lab board to stop acquiring data after a specified number of samples. Until you apply the stop trigger pulse at the EXTTRIG input, a data acquisition operation remains in a continuous acquisition mode, indefinitely writing and rewriting data to the buffer. You can select pretriggering by calling `DAQ_StopTrigger_Config`.
- A *timebase clock* is a clock signal that is the timebase for the sample-interval or scan-interval counter. Counter B0 is used to provide the timebase clock when the interval between samples or updates is greater than 65,535 µs. The timebase for sample and scan interval is the same.

See the *Lab-PC+ User Manual*, *SCXI-1200 User Manual*, *DAQPad-1200 User Manual*, or *DAQCard-1200 User Manual* for more information regarding these signals.

Lab Board Data Acquisition Rates

The maximum recommended data acquisition rates with the Lab boards and the recommended values for settling time versus gain are shown in Table 2-2. The required settling time depends on the gain setting used for each channel. Also, this settling time limits data acquisition rates.

Table 2-2. Typical Settling Accuracies for Maximum Multiple-Channel Scanning Rates for the Lab Boards

Gain	Maximum Acquisition Rate	Settling Time
1, 2, 5, 10, 20, 50	62.5 kS/s	16 µs
100	20 kS/s	50 µs

For the Lab boards, the settling accuracy is always ±0.5 LSB.

If you are using SCXI with your Lab board, refer to the *SCXI Data Acquisition Rates* section in Chapter 3, *Software Overview*, for the effect of SCXI module settling time on your DAQ device rates.

Lab Board Analog Output

The Lab board contains two analog output channels numbered 0 and 1. Each analog output channel contains a 12-bit DAC. The DACs are double buffered, which facilitates accurate waveform generation via the delayed update mode. You can hardware jumper configure each analog output channel for unipolar or bipolar voltage output on the Lab-PC+. You can software configure each analog output channel on the SCXI-1200, DAQPad-1200, and DAQCard-1200. An onboard voltage reference of +5 V is provided for the analog output channels.

Lab Board Waveform Generation

The Waveform Generation functions can continuously write values to either one or both analog output channels using an onboard or external clock to update the DACs at regular intervals. The values are contained in a buffer that you allocate and fill. The resultant voltages produced at the analog output channels depend on the value of the integer numbers in the buffer, the level of the reference voltage, and the polarity setting.

Lab Board Digital I/O

The Lab board contains 24 bits of digital I/O. These bits are divided into a set of three digital I/O ports of eight bits each. The Intel 8255A Parallel Peripheral Interface chip controls digital I/O on the Lab board. The digital I/O ports are labeled as ports PA, PB, and PC on the I/O connectors, as shown in your Lab board user manual.

You can configure all three ports as either input ports or output ports. These ports are referred to as ports 0, 1, and 2 for the Digital I/O functions, in which:

- port PA = port 0
- port PB = port 1
- port PC = port 2

Ports 0 and 1 support both latched (handshaking) and nonlatched (no-handshaking) modes. Port 2 supports nonlatched mode only. The digital lines making up port 2 (PC) are used as handshaking lines for both ports 0 and 1 whenever either is configured for latched mode;

therefore, port 2 is not available for Digital I/O functions whenever either port 0 or port 1 is configured for latched mode.

Using an SCXI chassis with a Lab-PC+, SCXI-1200, or DAQCard-1200 renders port PB unavailable. NI-DAQ also reserves line 0 of port PC for input from the SCXI hardware if you have SCXI configured. The remaining lines of port PC are available for input only.

Lab Board Groups

You can group any combination of ports 0 and 1 on the Lab board together to make up larger ports.

Lab Board Interval Counter/Timer Operation

Figure 2-5 diagrams the 16-bit counters available on the Lab board.

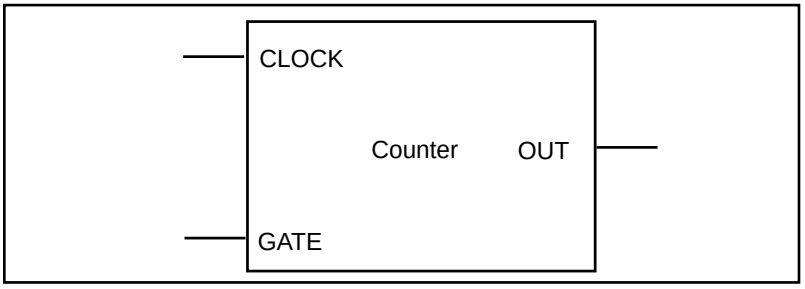


Figure 2-5. Interval Counter Block Diagram

Each counter has a clock input, a gate input, and an output. You can use a counter to count the falling edges of the signal applied to the CLK input. The Lab board uses the counter gate input to gate counting operations. Refer to the counter data sheet included in your device user manual to see how the gate inputs affect the counting operation in different counting modes.

Lab Board Counter/Timers

The Lab board contains two onboard 8253 Programmable Interval Timer chips that have three independent 16-bit counter/timers. One of these chips, the 8253-A, is reserved for data acquisition and waveform generation operations. You can use the three counters on the other chip,

the 8253-B, for counting/timing operations. NI-DAQ uses the three counter/timers from the 8253-B as follows:

- Counter 0 is used for extending the timebase for data acquisition or waveform generation when the interval between samples or updates is greater than 65,535 µs.
- Counter 1 can be reserved for data acquisition using interval scanning on all Lab boards or Track*/Hold manipulation for the SCXI-1140 with the Lab-PC+, SCXI-1200, and DAQCard-1200. Counter 1 is otherwise available for counting/timing operations.
- Counter 2 is always available.

Figure 2-6 shows the connections of the 8253-B Counter/Timer signals to the Lab board I/O connector.

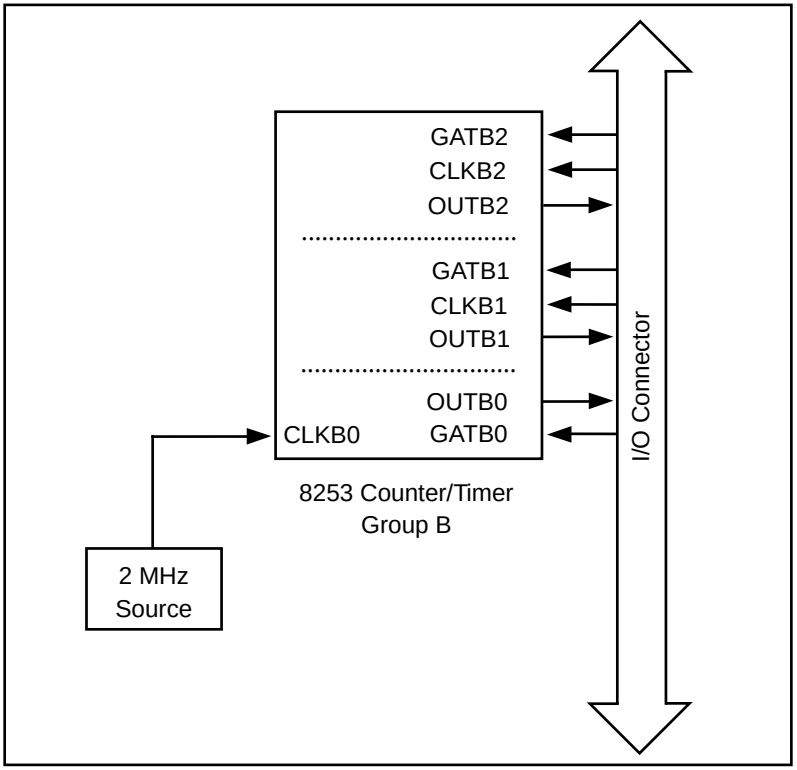


Figure 2-6. Lab Board Counter/Timer Signal Connections

Each counter has a clock input, a gate input, and an output labeled CLK, GAT, and OUT, respectively. The CLK pin for counters B1 and B2 and the GATE and OUT pins for counter B0, B1, and B2 are on the Lab board I/O connector.

SCXI-1200, DAQPad-1200, and DAQCard-1200 Performance

All I/O operations for the SCXI-1200, DAQPad-1200, and DAQCard-1200 are interrupt driven.

☞ **Note:** ***Because the SCXI-1200 and DAQPad-1200 communicate with the PC via the parallel port, your acquisition gates will be limited by the type of parallel port you have in your system (either a standard Centronics port or an Enhanced parallel port). Because both devices have a 2,048 word deep FIFO, you can acquire bursts of less than 2,048 samples at maximum speed.***

As with all interrupt-driven operations, you can exceed the ability of your system to handle the interrupt traffic if you increase your analog input sampling rate or analog output update rate beyond your system's capability. To determine the limits of your system, you can start at lower rates and gradually increase them until your computer begins to appear sluggish.

DAQPad-1200 Printing

The DAQPad-1200 is equipped with a pass-through parallel port for connection to a printer. To print with this configuration, the DAQPad-1200 must be in an idle state. The DAQPad-1200 is idle when it is not generating any interrupts. In other words, do not initiate an asynchronous analog I/O or digital I/O operation and try to print while these operations are ongoing.

SCXI-1200, DAQPad-1200, and DAQCard-1200 Calibration

The SCXI-1200, DAQPad-1200, and DAQCard-1200 are shipped with factory-calibration coefficients stored in an EEPROM. If you want to recalibrate either unit, use the `Calibrate_1200` function.

The PC-LPM-16 Multifunction I/O Board

PC-LPM-16 Analog Input

The PC-LPM-16 contains 16 multiplexed, single-ended analog input channels numbered 0 through 15. The analog input channels are driven into a 12-bit, self-calibrating ADC. The PC-LPM-16 has no gains on the analog input.

You can hardware jumper configure analog input on the PC-LPM-16 for four different input ranges:

- 0 to +5 V (unipolar)
- 0 to +10 V (unipolar)
- -2.5 to +2.5 V (bipolar)
- -5 to +5 V (bipolar)

You can initiate A/D conversions through software or by applying active low pulses to the EXTCONV* input on the PC-LPM-16 I/O connector. A 16-word-deep FIFO memory on the board stores up to 16 A/D conversion results.

PC-LPM-16 Data Acquisition

The PC-LPM-16 can perform single-channel data acquisition and multiple-channel scanned data acquisition. For single-channel data acquisition, you select a single analog input channel. The board performs a single A/D conversion on that channel every sample interval.

For multiple-channel scanned data acquisition, the PC-LPM-16 scans a sequence of analog input channels. A sample interval indicates the time to elapse between A/D conversions on each channel in the sequence. You only need to select a single starting channel to select the sequence of channels to scan. The board then scans the channels in consecutive order until channel 0 is reached and the scan begins anew with the starting channel. If the starting channel is channel 3, for example, the scan sequence is as follows:

channel 3, channel 2, channel 1, channel 0, channel 3, and so on

You can use both the single-channel and multiple-channel acquisitions with the double-buffered mode. Double-buffered mode fills the user-specified buffer continuously. You can call the `DAQ_DB_Transfer`

to transfer old data into a second buffer before overwriting the old data with new data. NI-DAQ transfers data out of one half of the buffer while filling the other half with new data.

You can use SCXI modules as a data acquisition front end for the PC-LPM-16 to signal condition the input signals and multiplex the channels. You can use all the modes just described in conjunction with SCXI for single-channel acquisitions; however, multiple-channel acquisitions are only supported when using the SCXI-1120 or SCXI-1121 modules in Parallel mode. *The SCXI Hardware* section later in this chapter describes how the SCXI functions are used to set up the SCXI modules for a data acquisition to be performed by a DAQ board.

PC-LPM-16 Data Acquisition Timing

Timing for data acquisition can be performed by the onboard MSM82C53 counter/timer or externally. The MSM82C53 Counter/Timer has three independent 16-bit counters/timers, which are assigned as follows:

- Counter 0 is a sample-interval counter for data acquisition that is available if no data acquisition is in progress.
- Counter 1 is available for general-purpose counting functions.
- Counter 2 is available for general-purpose counting functions.

Data acquisition timing involves the following timing signals:

- A *conversion pulse* is a signal that generates a pulse once every sample interval, causing the board to initiate an A/D conversion. This signal can be generated by the onboard, programmable sample-interval clock supplied by the MSM82C53 Counter/Timer on the PC-LPM-16, or can be supplied externally through the I/O connector EXTCONV* input. You can select external conversion pulses by calling `DAQ_Config`. If you do not want to use external conversion pulses, you should disconnect the EXTCONV* pin on the I/O connector to prevent extra conversions.
- A *timebase clock* is a clock signal that is the timebase for the sample-interval counter. Counter 0 of the MSM82C53 uses a 1 MHz clock as its timebase.

See the *PC-LPM-16 User Manual* for more information regarding these signals.

PC-LPM-16 Data Acquisition Rates

Table 2-3 shows the maximum recommended data acquisition rates for the PC-LPM-16.

Table 2-3. Maximum Recommended Data Acquisition Rates for the PC-LPM-16

Sample Mode	Maximum Acquisition Rate	Sample Interval
Single-channel scan	50 kS/s	20 µs
Multiple-channel scan (except 0 to 10 V input range)	50 kS/s	20 µs
Multiple-channel scan (0 to 10 V input range)	50 kS/s (typical) 45 kS/s (worst case)	20 µs 22 µs

If you are using SCXI with your DAQ device, refer to the *SCXI Data Acquisition Rates* section in Chapter 3, *Software Overview*, for the effect of SCXI module settling time on your DAQ device rates.

PC-LPM-16 Digital I/O

The PC-LPM-16 board contains one 8-bit digital input port and one 8-bit digital output port. The digital I/O ports are labeled DIN and DOUT on the I/O connector, as shown in the *PC-LPM-16 User Manual*. The ports are referred to as ports 0 and 1 for the Digital I/O functions, in which:

- DOUT = port 0
- DIN = port 1

You can program ports 0 and 1 for nonlatched (no-handshaking) mode only. You can use port 0 for nonlatched digital output mode. You can use port 1 for nonlatched digital input mode.

Using an SCXI chassis with a PC-LPM-16 renders lines 4, 5, 6, and 7 of port 0 and line 6 of port 1 unavailable.

PC-LPM-16 Interval Counter/Timer Operation

The PC-LPM-16 interval counter/timer operation is the same as for the Lab-PC+, except that the PC-LPM-16 has a single counter. Refer to the *Lab Board Interval Counter/Timer Operation* section earlier in this chapter for detailed information.

PC-LPM-16 Counter/Timers

The PC-LPM-16 contains an onboard MSM82C53 Programmable Interval Timer chip that has three independent 16-bit counter/timers. NI-DAQ uses the three counter/timers from the 82C53 as follows:

- Counter 0 is used for data acquisition operations.
- Counter 1 is available for counting/timing operations.
- Counter 2 can be reserved for Track*/Hold manipulation for the SCXI-1140, and is otherwise available for counting/timing operations.

Figure 2-7 shows the connections of the MSM82C53 Counter/Timer signals to the PC-LPM-16 I/O connector.

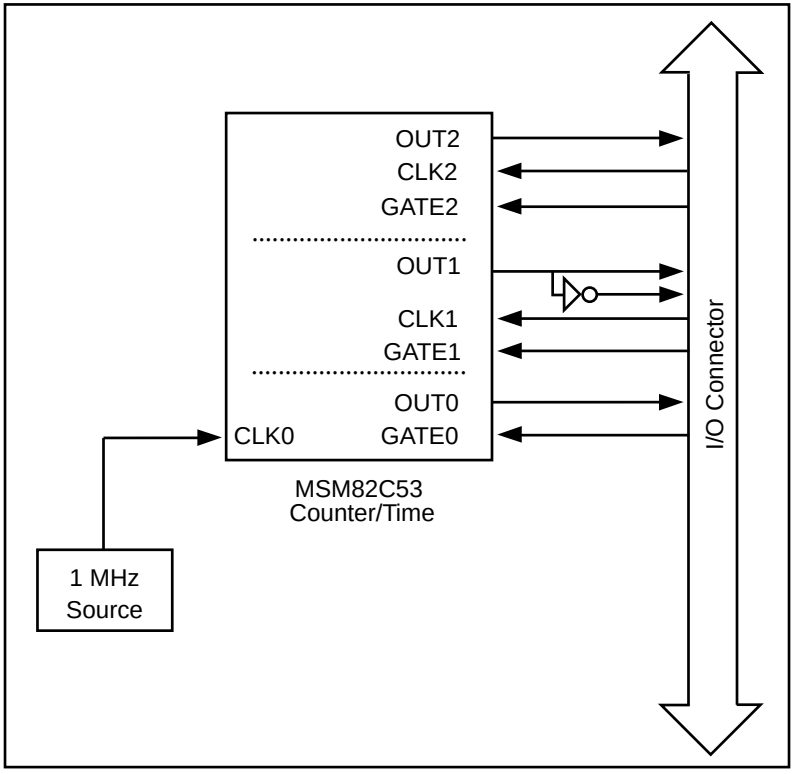


Figure 2-7. PC-LPM-16 Counter/Timer Signal Connections

The counter has a clock input, a gate input, and an output labeled CLK, GATE, and OUT, respectively. The CLK pin for counters 1 and 2 and

the GATE and OUT pins for counters 1, 2, and 3 are available on the PC-LPM-16 I/O connector. The inverted OUT1 signal is also available on the I/O connector.

The DAQCard-500 and DAQCard-700 Multifunction I/O Devices

DAQCard-500 and DAQCard-700 Analog Input

The DAQCard-500 provides 8 multiplexed, single-ended analog input channels. The DAQCard-700 provides 16 multiplexed, single-ended or eight multiplexed differential analog input channels. The analog input channels for both are driven into a 12-bit ADC. Neither device has gains on the analog input.

You can configure the DAQCard-700 analog input for three different bipolar input ranges:

- -2.5 to +2.5 V
- -5 to +5 V
- -10 to +10 V

You can configure the DAQCard-500 for only the -5 to +5 V range.

You can initiate A/D conversions through software or by applying active low pulses to the EXTCONV* input on the device I/O connector. A 512-word-deep FIFO memory on the DAQCard-700 stores up to 512 A/D conversion results. On the DAQCard-500, a 16-word-deep FIFO stores up to 16 A/D conversion results.

DAQCard-500 and DAQCard-700 Data Acquisition

The DAQCard-500 and DAQCard-700 can perform single-channel data acquisition and multiple-channel scanned data acquisition. For single-channel data acquisition, you select a single analog input channel. The device performs a single A/D conversion on that channel every sample interval.

For multiple-channel scanned data acquisition, the device scans a sequence of analog input channels. A sample interval indicates the time to elapse between A/D conversions on each channel in the sequence. You need only to select a single starting channel to select the sequence

of channels to scan. The device then scans the channels in consecutive order until channel 0 is reached and the scan begins anew with the starting channel. If the starting channel is channel 3, for example, the scan sequence is as follows:

channel 3, channel 2, channel 1, channel 0, channel 3, and so on

You can use both the single-channel and multiple-channel acquisitions with the double-buffered mode. Double-buffered mode fills the user-specified buffer continuously. You can call the `DAQ_DB_Transfer` to transfer old data into a second buffer before overwriting the old data with new data. NI-DAQ transfers data out of one half of the buffer while filling the other half with new data.

You can use SCXI modules as a data acquisition front end for the DAQCard-700 to signal condition the input signals and multiplex the channels. You can use all the modes just described in conjunction with SCXI for single-channel acquisitions; however, multiple-channel acquisitions are only supported when using the SCXI-1120 or SCXI-1121 modules in Parallel mode. *The SCXI Hardware* section later in this chapter describes how the SCXI functions are used to set up the SCXI modules for a data acquisition to be performed by a DAQ device.

You cannot use the DAQCard-500 with SCXI.

DAQCard-500 and DAQCard-700 Data Acquisition Timing

Timing for data acquisition can be performed by the onboard MSM82C53 Counter/Timer or externally. The MSM82C53 Counter/Timer has three independent 16-bit counters/timers, which are assigned as follows:

- Counter 0 is a sample-interval counter for data acquisition that is available if no data acquisition is in progress.
- Counter 1 is available for general-purpose counting functions.
- Counter 2 is available for general-purpose counting functions.

Data acquisition timing involves the following timing signals:

- A *conversion pulse* is a signal that generates a pulse once every sample interval, causing the device to initiate an A/D conversion. This signal can be generated by the onboard, programmable sample-interval clock supplied by the MSM82C53 Counter/Timer, or can be supplied externally through the I/O connector EXTCONV* input. You can select external conversion pulses by

calling `DAQ_Config`. If you do not want to use external conversion pulses, you should disconnect the EXTCONV* pin on the I/O connector to prevent extra conversions.

- A *timebase clock* is a clock signal that is the timebase for the 8254 sample-interval counter. Counter 0 of the MSM82C53 uses a 1 MHz clock as its timebase.

See your device user manual for more information regarding these signals.

If you are using SCXI with your DAQCard-700, refer to the *SCXI Data Acquisition Rates* section later in this chapter for the effect of SCXI module settling time on your DAQ device rates.

DAQCard-500 and DAQCard-700 Digital I/O

The DAQCard-500 has one 4-bit digital input port and one 4-bit digital output port. The DAQCard-700 contains one 8-bit digital input port and one 8-bit digital output port. The digital I/O ports are labeled DIN and DOUT on the I/O connector, as shown in the appropriate device user manual. The ports are referred to as ports 0 and 1 for the Digital I/O functions, in which:

- DOUT = port 0
- DIN = port 1

You can program ports 0 and 1 for nonlatched (no-handshaking) mode only. You can use port 0 for nonlatched digital output mode. You can use port 1 for nonlatched digital input mode.

Using an SCXI chassis with the DAQCard-700 renders digital lines 4, 5, 6, and 7 of port 0 and line 6 of port 1 unavailable.

DAQCard-500 and DAQCard-700 Interval Counter/Timer Operation

The DAQCard-500 and DAQCard-700 interval counter/timer operation is the same as for the Lab-PC+, except that the DAQCard-500 and DAQCard-700 has a single counter chip. Refer to the *Lab Board Interval Counter/Timer Operation* section earlier in this chapter for detailed information.

DAQCard-500 and DAQCard-700 Counter/Timers

The DAQCard-500 and DAQCard-700 contain an onboard MSM82C53 Programmable Interval Timer chip that has three

independent 16-bit counter/timers. NI-DAQ uses the three counter/timers from the 82C53 as follows:

- Counter 0 is used for data acquisition operations.
- Counter 1 is available for counting/timing operations.
- Counter 2 can be reserved for Track*/Hold manipulation for the SCXI-1140 (with the DAQCard-700 only), and is otherwise available for counting/timing operations.

Figure 2-8 shows the connections of the MSM82C53 Counter/Timer signals to the DAQCard-500 and DAQCard-700 I/O connector.

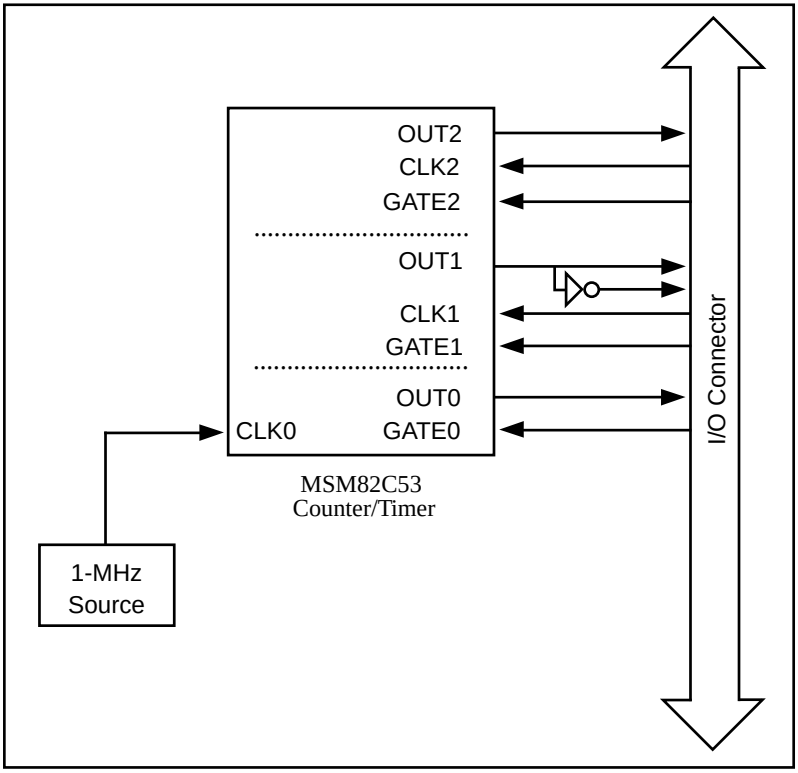


Figure 2-8. DAQCard-500 and DAQCard-700 Counter/Timer Signal Connections

☞ **Note:** ***The inverted OUT1 signal is NOT available only on the DAQCard-500.***

The counter has a clock input, a gate input, and an output labeled CLK, GATE, and OUT, respectively. The CLK pin for counters 1 and 2 and

the GATE and OUT pins for counters 1, 2, and 3 are available on the device I/O connector. The inverted OUT1 signal is also available on the DAQCard-700 I/O connector.

The AT-AO-6/10 Analog Output Boards

AT-AO-6/10 Analog Output

The AT-AO-6 and AT-AO-10 contain six or 10 analog output channels, respectively. Each analog output channel contains a 12-bit DAC. The DACs are double buffered, which facilitates accurate waveform generation via the delayed update mode. You can hardware jumper configure each analog output channel for unipolar or bipolar voltage output. An onboard voltage of +10 V or an externally connected voltage signal is available for analog output channel pairs as a voltage reference. There is a 4 to 20 mA current output associated with each analog output channel. You control the current output by writing voltages to the DACs with `AO_Write` or `AO_VWrite`. See the *AT-AO-6/10 User Manual* for the voltage/current relationship.

AT-AO-6/10 Waveform Generation

The Waveform Generation functions can continuously write values to any number of analog output channels using an onboard or external clock to update the DACs at regular intervals. You can assign output channels to one of two groups. Each group has its own dedicated update clock. The values written to the DACs are contained in a buffer that you allocate and fill. The resultant voltages produced at the analog output channels depend on the value of the integer numbers in the buffer, the level of the reference voltage, and the polarity setting.

Waveform Generation Using Onboard Memory

The AT-AO-6/10 supports FIFO mode waveform generation. In this mode, the board transfers the values for one cycle of waveform to onboard DAC FIFO memory only once. Then the board cycles through these values to generate continuous waveform. Thus, no interrupt service or DMA operation is required to transfer more data to the FIFO.

The following conditions must be satisfied to use FIFO mode waveform generation:

- The waveform buffer fits in the DAC FIFO.

- Double-buffered waveform generation mode is disabled.
- The number of iterations is equal to 0.

Notice that this mode is available for group 1 only.

Hardware Restrictions for the AT-AO-6/10

The following hardware restriction applies to the AT-AO-6/10:

- You cannot split channel pairs between groups (channel pairs are 0 and 1, 2 and 3, 4 and 5, and so on). For example, you can assign channel 4 alone to group 1, but you cannot then assign channel 5 to group 2.

AT-AO-6 group 1 assignments are as follows:

- 0 to *n*, where $n \leq 5$ and the channel list is consecutive, or any one channel.
- Uses interrupts/DMA with FIFO.
- Interrupt when the FIFO is half full; thus, group 1 will be faster than group 2, even when interrupts are used for both.
- If more than one channel is in the channel list, then channel 0 must be the first channel in that list.

AT-AO-6 group 2 assignments are as follows:

- channels 0 or 1 cannot be in group 2.
- Uses interrupts only.

AT-AO-10 group assignments are as follows:

- All rules of assignment for the AT-AO-6 apply to the AT-AO-10.
- 0 to *n*, where $n \leq 9$ and the channel list is consecutive, or any one channel.
- If exactly one channel is assigned to group 1, it cannot be channel 8 or 9.

AT-AO-6/10 Digital I/O

The AT-AO-6/10 contains eight bits of digital I/O. These bits are divided into a set of two digital I/O ports of four bits each. The 4-bit

digital I/O ports are labeled as ports DIOA and DIOB. These ports are referred to as ports 0 and 1 by the Digital I/O functions, in which:

- port DIOA = port 0
- port DIOB = port 1

You can configure port 0 or 1 as either an input or an output port. Any port that you configured as an output port has read-back capability (that is, by reading the port, you can determine what digital value the output port is currently asserting). The AT-AO-6/10 digital I/O ports operate in nonlatched mode only.

The EISA-A2000 Analog Input Board

EISA-A2000 Analog Input

The EISA-A2000 contains four simultaneously sampled, single-ended analog input channels numbered 0 through 3. Each analog input channel has a sample-and-hold circuit. The EISA-A2000 samples one, two, or four input channels simultaneously. The board multiplexes these input channels into a single unity gain stage followed by a 1 µs conversion time, 12-bit resolution ADC, which reads and converts each selected channel in turn.

The signal range of each input channel is ±5 V when you select DC coupling and ±5 V peak AC with ± 25 VDC offset when you select AC coupling.

You can initiate A/D conversions through software or by applying active-low pulses to the SAMPCLK* input on the EISA-A2000 I/O connector or active high pulses to the CLOCKI RTSI bus input. The 512-word-deep FIFO memory on the board stores up to 512 A/D conversion results.

The channels that you can select are as follows:

- One channel—channel 0, 1, 2, or 3
- Two channels—channels 0 and 1, or 2 and 3
- Four channels—channels 0 through 3

The EISA-A2000 operates exactly the same way whether one or many channels are sampled.

EISA-A2000 Data Acquisition

The EISA-A2000 operates in several trigger modes for data acquisition—pretrigger mode, posttrigger mode, or posttrigger mode with delay. In pretrigger mode, the EISA-A2000 acquires a programmed number of samples both before and after the board receives a trigger. In posttrigger mode, the board acquires a programmed number of samples after the trigger. In posttrigger mode with delay, the EISA-A2000 waits to acquire samples until a programmed time interval has elapsed after the board receives the trigger.

The EISA-A2000 has two main trigger sources—analog or digital. The board can receive the analog trigger from any one of the input channels or the ATRIG input on the I/O connector. Analog trigger circuitry causes a trigger when the selected input channels reach a preprogrammed slope and level. The board can receive a rising or falling edge digital trigger from the DTRIG I/O connector input. Alternatively, the board can receive digital triggers over the RTSI bus. After you configure the board to acquire samples, the EISA-A2000 can trigger and acquire data each time the board receives a trigger without being stopped or reprogrammed. This is called *multiple-frame data acquisition*, in which a frame is the data acquired with each trigger.

☞ **Note:** ***You cannot use SCXI with the EISA-A2000.***

EISA-A2000 Data Acquisition Timing

Timing for data acquisition is performed by the onboard Am9513 Counter/Timer or by the external sample clock, SAMPCLK* on the I/O connector or CLOCKI on the RTSI bus. Data acquisition timing involves the following timing signals and counters:

- The *trigger* is a signal that the EISA-A2000 that generates locally or receives from the SAMPLCLK* I/O connector input or from the RTSI bus. This signal determines when posttrigger sampling begins.
- The *sample clock* is a signal that the EISA-A2000 generates locally or receives from the SAMPLCLK* I/O connector input or from the RTSI bus.
- The *sample interval counter* is an EISA-A2000 counter that generates the onboard sample clock.

- The *sample counter* is the EISA-A2000 counter that counts posttrigger scans (multiple-channel samples) and stops acquisition when the board has acquired the programmed number.
- The *delay counter* is the EISA-A2000 counter that counts the specified time delay after the trigger and then starts posttrigger acquisition when time expires.
- The *timebase* is the onboard clock sources for the sample-interval and delay counters. Available timebases include 5 MHz, 1 MHz, 100 kHz, 10 kHz, 1 kHz, and 100 Hz. In addition, you can supply external timebase clocks through the RTSI switch signals SOURCE2 and GATE2.

The board can receive additional timing signals from the RTSI bus. See the *EISA-A2000 User Manual* for more information regarding these signals.

EISA-A2000 Data Acquisition Rates

The maximum data acquisition rate for the EISA-A2000 is 1 µs/channel; in other words, 1 µs, 2 µs, or 4 µs for one, two, or four channels, respectively (see Table 2-4). Converting at a rate faster than the maximum data acquisition rate causes the board to miss points, and the data returned is an inaccurate representation of the signal being measured. In Windows 3.1, your maximum rate may be lower due to greater interrupt latencies and more memory fragmentation in this environment. Setting **NIDirectDMAProgramming** to yes in the Windows `SYSTEM.INI` file increases your maximum rate.

Table 2-4. Maximum EISA-A2000 Data Acquisition Rates

Number of Channels	Maximum Data Acquisition Rate
1	1 µs (1 MS/s)
2	2 µs (500 kS/s)
4	4 µs (250 kS/s)

EISA-A2000 Counter/Timer Operation

The EISA-A2000 counter/timer operation is the same as for the MIO-16. Refer to *Am9513-Based Device Counter/Timer Operation* earlier in this chapter for detailed information.

EISA-A2000 Counter/Timers

The EISA-A2000 has one unused 16-bit counter/timer from the onboard Am9513 System Timing chip. This counter, counter 2, is made available for general use via the RTSI bus. See *The RTSI Bus Trigger Functions* section of Chapter 3, *Software Overview*, for more information. The five counter/timers are used in NI-DAQ as follows:

- Counter 1 is always reserved for pretrigger hold off and posttrigger delay.
- Counter 2 is available for general-purpose counting functions.
- Counter 3 is a scan-interval counter that is always reserved for data acquisition.
- Counter 4 is a sample counter that is always reserved for data acquisition.
- Counter 5 is a sample counter that is always reserved for data acquisition.

The GATE2, SOURCE2, and OUT2 signals are connected to the RTSI bus via the onboard RTSI switch as shown in Figure 2-9.

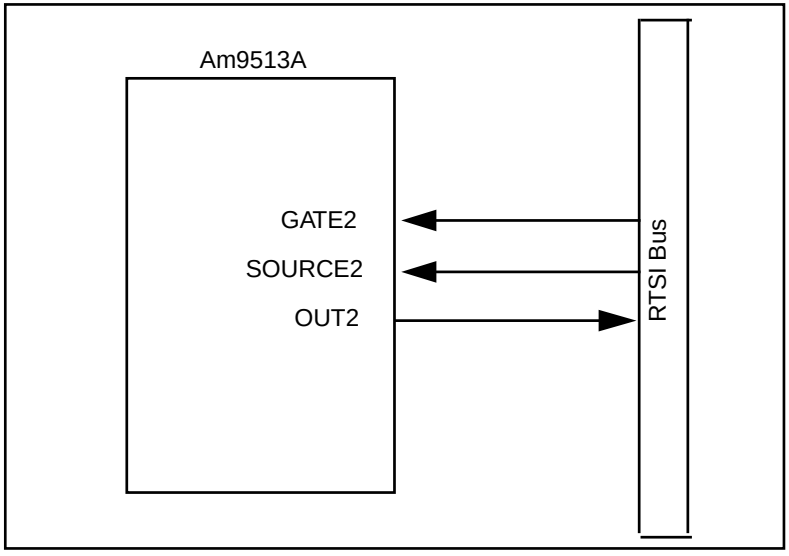


Figure 2-9. EISA-A2000 Counter/Timer Signal Connections

The AT-A2150 Audio Input Board

There are two versions of the AT-A2150—the AT-A2150C and the AT-A2150S. Each version has a different set of sampling frequencies. However, they are software compatible and can be programmed the same way using NI-DAQ function calls. AT-A2150 refers to both versions unless otherwise specified.

AT-A2150 Analog Input

The AT-A2150 has four simultaneously sampled, single-ended analog input channels. Oversampling delta-sigma modulating ADCs produce 16-bit resolution readings.

The signal range of each analog input channel is ±2.828 V when you select DC coupling, and ±2.828 VAC with a maximum input voltage of ±10 VDC when you select AC coupling.

The AT-A2150 continuously samples and produces 16-bit results at the current sampling rate. Reading(s) returned by the `MAI_Read` function are the most recently acquired sample(s).

Although you can synchronize an AT-A2150 to use the timing signals of another AT-A2150, other external clock signals are not supported. To synchronize multiple AT-A2150s, refer to *The RTSI Bus Trigger Functions* section of Chapter 3, *Software Overview*.

AT-A2150 Data Acquisition

The AT-A2150 has four simultaneously sampled, single-ended analog input channels (ACH) numbered 0 through 3. The combination of channels that you can select are as follows:

- One channel—ACH0, ACH1, ACH2, or ACH3
- Two channels—ACH0 and ACH1, or ACH2 and ACH3
- Four channels—ACH0, ACH1, ACH2, and ACH3

The signal range of each analog input channel is ±2.828 V when you select DC coupling, and ±2.828 VDC with a maximum input voltage of ±20 V when you select AC coupling.

The AT-A2150 can operate in several trigger modes—pretrigger mode, posttrigger mode, posttrigger mode with delay, or software posttrigger mode. In pretrigger mode, the AT-A2150 acquires a programmed

number of samples both before and after the board receives a trigger. In posttrigger mode, the AT-A2150 acquires a programmed number of samples after the trigger. In posttrigger mode with delay, the AT-A2150 waits a specified amount of time after the trigger before acquiring data. The software posttrigger mode is a special case of the posttrigger mode in which the software acts as the trigger and the board acquires data immediately following the `MDAQ_Start` call.

There are two hardware trigger sources—analog or digital. The board can receive the analog trigger from any one of the input channels. Analog trigger circuitry causes a trigger when the selected input channel reaches a specified slope and level. To eliminate false triggering due to noise in the analog trigger signal, use `Trigger_Window_Config` to specify a hysteresis window. If you want a digital trigger, you can apply the trigger to the digital trigger I/O connector. When using a digital trigger, notice that the first sample after the trigger does not appear in the A/D FIFO (also the acquisition buffer) until 34 or 35 samples (depending on the time of the trigger) from each channel being sampled have been acquired. This means that in the posttrigger mode, you have 34 or 35 samples of pretrigger data in the acquisition buffer. This behavior is a feature of the digital anti-aliasing filter. Alternatively, the board can receive digital triggers over the RTSI bus. After you have configured the board to acquire samples, the AT-A2150 can trigger and acquire data each time the board receives a trigger without being stopped or reprogrammed. This is called multiple-frame data acquisition, in which a frame is the data acquired with each trigger.

The AT-DSP2200 Digital Signal Processing Board

AT-DSP2200 Analog Input

The AT-DSP2200 has two simultaneously sampled, single-ended analog input channels. Oversampling delta-sigma modulating ADCs produce 16-bit resolution readings.

The signal range of each analog input channel is ±2.828 V when you select DC coupling, and ±2.828 VAC with a maximum input voltage of ±10 VDC when you select AC coupling.

The AT-DSP2200 continuously samples and produces 16-bit results at the current sampling rate. Reading(s) returned by the `MAI_Read` function are the most recently acquired sample(s).

Although you can synchronize an AT-DSP2200 to use the timing signals of another AT-DSP2200 or AT-A2150, other external clock signals are not supported. To synchronize multiple boards, refer to *The RTSI Bus Trigger Functions* section of Chapter 3, *Software Overview*.

AT-DSP2200 Data Acquisition

The AT-DSP2200 has two simultaneously sampled, single-ended analog input channels (ACH) numbered 0 and 1. The combination of channels that you can select are as follows:

- One channel—ACH0 or ACH1
- Two channels—ACH0 and ACH1

Use the `MAI` and `MDAQ` functions to acquire analog input data with the AT-DSP2200. You can use the `DSP2200_Config` function (see Chapter 2, *Function Reference*, of the *NI-DAQ Function Reference Manual for PC Compatibles*) to enable on-the-fly, floating-point conversion and demultiplexing. You can allocate data buffers on the AT-DSP2200 itself by using the `NI_DAQ_Mem_Alloc` call or the digital signal processing (DSP) memory management functions in the NI-DSP function library. See the *NI-DAQ Function Reference Manual for PC Compatibles* or the *NI-DSP Software Reference Manual for DOS/LabWindows* for details.

The AT-DSP2200 can operate in several trigger modes—pretrigger mode, posttrigger mode, posttrigger mode with delay, or software posttrigger made. In pretrigger mode, the board acquires a programmed number of samples both before and after the board receives a trigger. In posttrigger mode, the board acquires a programmed number of samples after the trigger. In posttrigger mode with delay, the board waits a specified amount of time after the trigger before the board acquires data. The software posttrigger mode is a special case of the posttrigger mode where the software acts as the trigger and the board acquires data immediately following the `MDAQ_Start` call.

There are two main hardware trigger sources—analog and digital. The board can receive the analog trigger from either of the input channels. Analog triggers occur when the selected input channel reaches a specified slope and level. To eliminate false triggering due to noise in the analog trigger signal, use the `Trigger_Window_Config`

function to specify a trigger hysteresis window. If you want a digital trigger, you can apply the trigger to the digital trigger I/O connector. When using a digital trigger, notice that the first sample after the trigger does not appear in the acquisition buffer until 34 or 35 samples (depending on the time of the trigger) from each channel being sampled have been acquired. This means that in the posttrigger mode, you have 34 or 35 samples of pretrigger data in the acquisition buffer. This behavior is a feature of the digital anti-aliasing filter. Alternatively, the board can receive digital triggers over the RTSI bus. After you configure the board to acquire samples, the AT-DSP2200 can trigger and acquire data each time the board receives a trigger without being stopped or reprogrammed. This is called multiple-frame data acquisition, in which a frame contains the data acquired with each trigger.

AT-DSP2200 Analog Output

The AT-DSP2200 contains two analog output channels numbered 0 and 1. Each channel contains a 16-bit delta-sigma modulating DAC. The DACs can generate voltages in the range of ±2.828 V. The `AO_Configure` and `AO_Update` functions do not support the AT-DSP2200.

AT-DSP2200 Waveform Generation

There are 16 fixed DAC update rates achieved by dividing one of four timebases by 1, 2, 4, or 8. With the Waveform Generation functions, the AT-DSP2200 can generate continuous analog waveforms. The waveform buffer can reside in PC memory or in onboard DSP memory. To use onboard DSP memory, you must allocate a buffer with the NI-DSP memory allocation function. You can use the DSP memory handle returned by this function as the buffer parameter in `WFM_Load`. The waveform buffer can contain 16-bit integer DAC values, floating-point DAC values, or floating-point voltages. Refer to `DSP2200_Config` in Chapter 2, *Function Reference*, of the *NI-DAQ Function Reference Manual for PC Compatibles*, for information on using on-the-fly, floating-point translation and scaling.

NI-DAQ does not currently support double-buffered waveform generation and transfers when the waveform buffer is a DSP buffer.

Serial Data Link support:

- An AT-MIO-16X or AT-MIO-64F-5 can acquire data directly into a DSP memory buffer. To do this, connect the MIO device to the

DSP board with a RTSI cable and specify a DSP handle in the buffer parameter of the DAQ or SCAN function call. Use DSP2200_Config to specify data translation and demultiplexing options for these operations.

- Currently, you cannot directly generate waveforms from a DSP buffer on an MIO device using serial data link.

☞ **Note:** ***When acquiring data and generating a waveform at the same time, only one of the buffers can be in PC memory. Both buffers can be DSP buffers.***

☞ **Note:** ***If you are using an AT-DSP2200 with LabWindows/CVI, you should install your NI-DSP for DOS disks to obtain the files*** DSP2200.OUT ***and*** DSP2200S.OUT***. These files are necessary to configure the AT-DSP2200 in the*** WDAQCONF ***configuration utility. None of the other files installed with NI-DSP for DOS will be used and can be deleted after installation.***

The PC-TIO-10 Timing I/O Board

PC-TIO-10 Counter/Timer Operation

The PC-TIO-10 counter/timer operation is the same as for the MIO-16. Refer to *Am9513-Based Device Counter/Timer Operation* earlier in this chapter for detailed information.

PC-TIO-10 Counter/Timers

The PC-TIO-10 contains two Advanced Micro Devices (AMD) Am9513 System Timing Controller (STC) chips, each of which provides five independent 16-bit counter/timers and a 4-bit programmable frequency output, FOUT. Each of the STCs is connected to an onboard 1 MHz frequency source, thus giving internal frequencies as specified previously. In addition, the SOURCE5 and SOURCE10 inputs are connected to a 5 MHz frequency source that yields increased timing resolution.

All counter/timers are available for general-purpose counting functions. All 10 counters and both FOUT outputs are connected to the I/O connector of the PC-TIO-10. The GATE, SOURCE, and OUTPUT lines of each of the counters except counters 5 and 10 are connected to the I/O connector. As mentioned previously, SOURCE5 and SOURCE10 are used as additional frequency inputs; consequently, only the GATE and OUTPUT lines of counters 5 and 10 are connected to the I/O connector.

Because the Am9513 integrated circuits operate independently, the next lower order counter of counter 1 is counter 5, and the next lower order counter of counter 6 is counter 10. Accordingly, the next higher order counter of counter 5 is counter 1, and the next higher order counter of counter 10 is counter 6.

PC-TIO-10 Digital I/O

The PC-TIO-10 board contains 16 bits of digital I/O. These bits are divided into a set of two digital I/O ports of eight bits each. The digital I/O ports are labeled as ports A and B on the I/O connector as shown in the *PC-TIO-10 User Manual*. These ports are referred to as ports 0 and 1 for the Digital I/O functions in which:

- port A = port 0
- port B = port 1

Digital I/O on this board is controlled by the Motorola MC6821 Peripheral Interface Adapter chip, the ports of which you can program on a bitwise basis—you can configure each bit individually for input or output. In addition, any bit that you configure as an output bit has read-back capability (that is, by reading the port associated with a particular bit, you can determine what digital value the output bit is currently asserting). The PC-TIO-10 digital I/O ports operate only in nonlatched mode.

The PC-DIO-96 Digital I/O Board

PC-DIO-96 Digital I/O

The PC-DIO-96 board contains 96 bits of digital I/O. These bits are divided into a set of 12 digital I/O ports of eight bits each. Digital I/O on this board is controlled by four Intel 8255A Parallel Peripheral Interface chips. The digital I/O ports are labeled as ports APA, APB, APC, BPA, BPB, BPC, CPA, CPB, CPC, DPA, DPB, and DPC on the I/O connector as shown in the *PC-DIO-96 User Manual*. You can configure all 12 ports as either input ports or output ports.

These ports are referred to as ports 0 through 11 for the Digital I/O functions in which:

- port APA = port 0
- port APB = port 1

- port APC = port 2
- port BPA = port 3
- port BPB = port 4
- port BPC = port 5
- port CPA = port 6
- port CPB = port 7
- port CPC = port 8
- port DPA = port 9
- port DPB = port 10
- port DPC = port 11

You can use ports 0, 1, 3, 4, 6, 7, 9, and 10 for both latched (handshaking) and nonlatched (no-handshaking) modes. You can use ports 2, 5, 8, and 11 only for nonlatched mode. The board uses the digital lines making up port 2 (APC) as handshaking lines for ports 0 and 1 whenever you configure either for latched mode; therefore, port 2 is not available for the Digital I/O functions whenever you configure either port 0 or port 1 for latched mode. The board uses the digital lines making up port 5 (BPC) as handshaking lines for ports 3 and 4 whenever you configure either for latched mode; therefore, port 5 is not available for the Digital I/O functions whenever you configure either port 3 or port 4 for latched mode. The board uses the digital lines making up port 8 (CPC) as handshaking lines for ports 6 and 7 whenever you configure either for latched mode; therefore, port 8 is not available for the Digital I/O functions whenever you configure either port 6 or port 7 for latched mode. The board uses the digital lines making up port 11 (DPC) as handshaking lines for ports 9 and 10 whenever you configure either for latched mode; therefore, port 11 is not available for the Digital I/O functions whenever you configure either port 9 or port 10 for latched mode.

PC-DIO-96 Groups

You can group any combination of ports 0, 1, 3, 4, 6, 7, 9, and 10 on the PC-DIO-96 to make up larger ports. For example, you can program ports 0, 3, 9, and 10 to make up a 32-bit handshaking port, or all eight ports to make up a 64-bit handshaking port. See *Digital I/O Application Hints* in Chapter 3, *Software Overview*, for more details.

The DIO-24, AT-MIO-16D, and AT-MIO-16DE-10 Digital I/O Devices

DIO-24, AT-MIO-16D, and AT-MIO-16DE-10 Digital I/O

The DIO-24, AT-MIO-16D, and AT-MIO-16DE-10 devices contain 24 bits of digital I/O. These bits are divided into a set of three digital I/O ports of eight bits each.

The digital I/O ports are labeled as ports PA, PB, and PC on the I/O connectors, as shown in your device user manual. You can configure all three ports as either input ports or output ports. The following table lists the port numbers to use in the Digital I/O functions.

Device	Port Names	Port Numbers
DIO-24	PA, PB, PC	0, 1, 2
AT-MIO-16DE-10	PA, PB, PC	2, 3, 4
AT-MIO-16D	PA, PB, PC	2, 3, 4

Ports PA and PB support both latched (handshaking) and nonlatched (no-handshaking) modes. You can use port PC only for nonlatched mode. The device uses the digital lines making up port PC as handshaking lines for both ports PA and PB whenever you configure either for latched mode; therefore, port PC is not available for Digital I/O functions whenever you configure either port PA or port PB for latched mode.

☞ **Note:** ***As discussed earlier in this chapter, the AT-MIO-16D contains eight additional bits of digital I/O split into two 4-bit ports called 0 and 1. The AT-MIO-16DE-10 also contains eight additional bits of digital I/O called port 0. The AT-MIO-16DE-10 does not have a port 1.***

DIO-24, AT-MIO-16D, and AT-MIO-16DE-10 Groups

You can group ports PA and PB together to make up a 16-bit port. See *Digital I/O Application Hints* in Chapter 3, *Software Overview*, for more details.

The DIO-32F Digital I/O Boards

DIO-32F Digital I/O

The DIO-32F contains 36 bits of digital I/O. These bits are divided into a set of four digital I/O ports of eight bits each, a 2-bit digital input port, and a 2-bit digital output port. The 8-bit digital I/O ports are labeled as ports DIOA, DIOB, DIOC, and DIOD on the I/O connector, as shown in the *AT-DIO-32F User Manual*. The 2-bit digital input port is labeled IN, and the 2-bit digital output port is labeled OUT on the I/O connector. These ports are referred to as ports 0 through 4 by the Digital I/O functions, in which:

- port DIOA = port 0
- port DIOB = port 1
- port DIOC = port 2
- port DIOD = port 3
- ports OUT and IN = port 4

You can configure ports 0 through 3 as either input ports or output ports. When you configure any of these ports as an output port, the port has read-back capability; that is, by reading the port, you can determine what digital value the output port is currently asserting. Port 4 is always configured for both input and output. Because the input and output pins of port 4 are physically separate, however, writing and then reading port 4 does not return the value written (unless, of course, OUT1 is wired to IN1 and OUT2 to IN2 at the I/O connector).

☞ **Note:** ***For AT Series boards of Revision B or earlier, you must configure ports 0 and 1 and ports 2 and 3 for operation in the same direction.***

You can also configure ports 0 through 3 for both latched and nonlatched modes. If you configure the ports for latched mode, you must assign the ports to one of two handshake groups. The DIO-32F I/O connector has handshake lines for the groups. These handshake lines are labeled REQ1 and REQ2 for request, and ACK1 and ACK2 for acknowledge. Signals received or generated on these handshake lines affect only the ports assigned to the group.

Port 4 is always configured as an I/O port. Writing to port 4 affects the output lines on the connector labeled OUT1 and OUT2. Reading from port 4 returns the digital value of the input lines on the connector

labeled IN1 and IN2. These lines are mapped to the bits of the byte written to and read from port 4 as follows:

Bit Number	Digital I/O Line Number
7 to 2	No significance
1	OUT2 IN2
0	OUT1 IN1 Least significant bit

You *cannot* configure port 4 for latched mode.

☞ **Note:** ***At least one DMA channel must be available for buffered operations (***`DIG_Block`*** and ***`DIG_DB`*** function calls).***

The AMUX-64T External Multiplexer Device

An AMUX-64T external multiplexer device expands the number of analog input signals that the MIO or AI device can measure. The AMUX-64T has 16 separate four-to-one analog multiplexer circuits. One AMUX-64T device can multiplex up to 64 analog input signals. You can cascade four AMUX-64T devices to permit up to 256 single-ended (128 differential) signals to be multiplexed by one MIO or AI MIO or AI device. Table 2-5 shows the number of channels available on the MIO or AI device with an external multiplexer.

Table 2-5. Analog Input Channel Range

Number of External Multiplexer (AMUX-64T) Devices	Channel Range	
	Single-Ended	Differential
0	0–15	0–7
1	0–63	0–31
2	0–127	0–31, 64–95
4	0–255	0–31, 64–95, 128–159, 192–223

`AI_Mux_Config` configures the number of multiplexer devices connected to the MIO or AI device. You then use the input channels in subsequent data acquisition calls with respect to the external AMUX-64T analog input channel numbers rather than the MIO or AI device onboard channel numbers. For example, with one external device, **channel** can have a value of 0 through 63 (single-ended), or 0 through 31 (differential). With two or four AMUX-64T devices, channel numbering of the second device can be from 64 through 127 (single-ended), or from 64 through 95 (differential). Therefore, single-ended and differential channels always begin at the same number on each device.

When you use more than one AMUX-64T device, address the channels on the different devices as follows:

AMUX-64T Device	Channel Number	
	Single-Ended	Differential
Device A	0–63	0–31
Device B	64–127	64–95
Device C	128–191	128–159
Device D	192–255	192–223

The channel address on each AMUX-64T depends on the switch setting on each device. See the *AMUX-64T User Manual* for more information on the external multiplexer device.

Scanning Order Using the AMUX-64T

The scanning counters on the AMUX-64T and on the MIO or AI device perform automatic scanning of the AMUX-64T analog input channels. When you perform a multiple-channel scanned data acquisition with an AMUX-64T, one of the counter/timers on the MIO or AI device normally available to you, counter 1, is used for switching the MIO or AI device onboard multiplexers.

Scanning is a simple operation for one AMUX-64T device but becomes more complex for multiple AMUX-64T devices. The following paragraphs explain in detail how channels are scanned from

the AMUX-64T. You must know this scanning order so that you can determine from which analog input channel the data was scanned during a data acquisition operation. When a single AMUX-64T device is connected to the MIO or AI device, you must scan four AMUX-64T input channels for every MIO or AI device channel. If two AMUX-64T devices are attached to the MIO or AI device, you must scan eight AMUX-64T channels for every MIO or AI device input channel. For example, channels 0 through 3 on AMUX-64T device A and channels 64 through 67 on AMUX-64T device B are multiplexed together into MIO or AI device channel 0. Notice that the MIO or AI device scans the first four channels on device A, followed by the first four channels on device B.

If four AMUX-64T devices are attached to the MIO or AI device, you must scan 16 AMUX-64T channels for every MIO or AI device input channel. For example, channels 0 through 3 on AMUX-64T device A, channels 64 through 67 on AMUX-64T device B, channels 128 through 131 on AMUX-64T device C, and channels 192 through 195 on device D are multiplexed together into MIO or AI device channel 0. Notice that the MIO or AI device scans the first four channels on device A, followed by the first four channels on device B, the first four channels on device C, and, finally, the first four channels on device D.

The order in which the MIO or AI device scans channels depends on the scan channel sequence specified in `SCAN_Setup`. This scan sequence is an array of MIO or AI device onboard channel numbers that indicates the order in which the MIO or AI device onboard channels are scanned. The scanning order on the AMUX-64T, however, is fixed. Table 2-6 shows the order in which the AMUX-64T

channels are scanned for every MIO or AI device input channel for different AMUX-64T configurations.

Table 2-6. AMUX-64T Scanning Order for Each MIO or AI Device Input Channel

MIO or AI Device Channel	AMUX-64T Channels						
	One Device	Two Devices		Four Devices			
	Device A	Device A	Device B	Device A	Device B	Device C	Device D
0	0–3	0–3	64–67	0–3	64–67	128–131	192–195
1	4–7	4–7	68–71	4–7	68–71	132–135	196–199
2	8–11	8–11	72–75	8–11	72–75	136–139	200–203
3	12–15	12–15	76–79	12–15	76–79	140–143	204–207
4	16–19	16–19	80–83	16–19	80–83	144–147	208–211
5	20–23	20–23	84–87	20–23	84–87	148–151	212–215
6	24–27	24–27	88–91	24–27	88–91	152–155	216–219
7	28–31	28–31	92–95	28–31	92–95	156–159	220–223
8	32–35	32–35	96–99	32–35	96–99	160–163	224–227
9	36–39	36–39	100–103	36–39	100–103	164–167	228–231
10	40–43	40–43	104–107	40–43	104–107	168–171	232–235
11	44–47	44–47	108–111	44–47	108–111	172–175	236–239
12	48–51	48–51	112–115	48–51	112–115	176–179	240–243
13	52–55	52–55	116–119	52–55	116–119	180–183	244–247
14	56–59	56–59	120–123	56–59	120–123	184–187	248–251
15	60–63	60–63	124–127	60–63	124–127	188–191	252–255

For example, if you use one AMUX-64T device, whenever NI-DAQ selects channel 0 on the MIO or AI device in the scan sequence, channels 0 through 3 on the AMUX-64T are automatically scanned. If

you use two AMUX-64T devices, channels 0 through 3 (device A) and channels 64 through 67 (device B) are automatically scanned whenever channel 0 is selected in the scan sequence. If you use four AMUX-64T devices, channels 0 through 3 (device A), channels 64 through 67 (device B), channels 128 through 131 (device C), and channels 192 through 195 (device D) are automatically scanned whenever channel 0 is selected in the scan sequence.

If you program the MIO or AI device with a sequential channel scan sequence of 0 through 7 or 0 through 15, the AMUX-64T channels are scanned from top to bottom in the order given in Table 2-6. Notice that if you use differential input configuration, you should enter only MIO or AI device channels 0 through 7 in the scan sequence in `SCAN_Setup`, in which case only the information pertaining to those channels in Table 2-6 applies. See the *AMUX-64T User Manual* for more information on the external multiplexer device.

☞ **Note:** ***The AT-MIO-64F-5 has onboard analog input channels that are numbered as follows:***

0 through 63, in single-ended mode
0 through 39, excluding 8 through 15, in differential mode

If you plan to use the AMUX-64T, the AMUX-64T channel numbers overlap with the onboard channels. For example, if you have one AMUX-64T device connected to the AT-MIO-64F-5, AMUX-64T channels 16 through 63 conflict with onboard single-ended channels (with the same numbers) on the analog input connector. Because of this overlapping of channel numbers, you must call the `MIO_Config` ***function whenever you want to use an AMUX-64T channel. For further details on the*** `MIO_Config` ***function, refer to Chapter 2,*** *Function Reference****, of the*** *NI-DAQ Function Reference Manual for PC Compatibles****.***

If you are using your MIO or AI device to power the AMUX-64T, you must be aware that not only must the fuses on the AMUX-64T be intact, but the fuse on the MIO or AI device must also work. All fuses are in working order when you receive them, but inadvertent contact between +5 V and ground can short the fuse.

The SC-2040 Track-and-Hold Accessory

The SC-2040 simultaneously amplifies and samples up to eight differential inputs and sends them in parallel as differential signals to

an E Series DAQ device. The SC-2040 has eight amplifiers, each with a DIP-switch programmable gain of 1, 10, 100, 200, 300, 500, 600, 700, or 800. The SC-2040 can be powered from either the DAQ device or from an external 5 V power supply. Scanning of the SC-2040 channels can be initiated through software or through an external trigger.

The track-and-hold circuitry acquires signals within 7 µs, and when hold mode is enabled, the signals settle within 1 µs. Then the E Series device samples the SC-2040 channels as fast as possible, after which the SC-2040 is put into track mode again.

If you are using DOS, before using the analog input functions to acquire data from your SC-2040, you must first call the `SC_2040_Configure` function. If you are using Windows and have configured your SC-2040 in `WDAQCONF`, you do not have to call `SC_2040_Configure` because NI-DAQ will retrieve the SC-2040 settings from the configuration file.

To take a single snapshot of voltages from several channels, you use the `SCAN_Setup` function followed by either `AI_Read_Scan` or `AI_VRead_Scan`. To repeatedly scan a group of channels, use the `SCAN_` series of functions.

The SC-2042-RTD Accessory

The SC-2042-RTD is a low-cost input device that connects directly to RTDs and provides current source excitation and RTD input signal routing to the Lab-PC+, MIO and AI devices, and E Series devices. The SC-2042-RTD has eight channels, each of which has one current excitation source (1 mA) and differential input signal routing to the DAQ device. The SC-2042-RTD works with RTDs from 10 Ω to 2 kΩ over their full temperature range, 4-wire ohms measurements of up to 6 kΩ, and overvoltage protection of 25 VDC.

To use the SC-2042-RTD, you must configure your DAQ device for differential mode.

The SC-2043-SG Accessory

The SC-2043-SG is a low-cost input device for bridge completion and calibration of strain gauge bridge circuits and input signal routing to the Lab-PC+, MIO and AI devices, and E Series devices. The

SC-2043-SG has eight channels; each channel has one-quarter or one-half (and full) bridge completion, a bridge-balancing circuit, and nonreferenced single-ended input signal routing to the DAQ device. There is one half-bridge reference common to all channels, with a jumper disable for full-bridge connections. The excitation supply is external and is common to all channels.

If you are using DOS, before using the analog input functions to acquire data from your strain gauge through the SC-2043-SG, you must call the `Set_DAQ_Device_Info` function. If you are using Windows and have configured your device in `WDAQCONF`, you do not have to call `Set_DAQ_Device_Info` because NI-DAQ will retrieve the SC-2043-SG settings from the configuration file.

To use the SC-2043-SG, you must configure your DAQ device for NRSE mode.

The SCXI Hardware

SCXI modules provide signal conditioning for analog input signals, isolation for analog and digital I/O signals, and channel multiplexing to increase the number of analog and digital signals provided by a DAQ device. You connect signals to your SCXI modules using shielded SCXI terminal blocks that plug into the front of the modules.

You can use a plug-in DAQ device in your computer to control the SCXI modules and perform A/D conversions on the conditioned analog input signals. You use an SCXI ribbon cable assembly to connect one of the SCXI modules in your chassis to the plug-in DAQ device in your PC.

You can also use the SCXI-1200 data acquisition and control module in your SCXI chassis to control the other SCXI modules and perform A/D conversions remotely at the SCXI chassis. NI-DAQ uses the parallel port on your PC to communicate with the SCXI-1200. You only need to cable the SCXI-1200 module to the PC parallel port, you do not need to cable anything to other modules in the chassis.

NI-DAQ supports the following DAQ devices for use with SCXI:

- MIO and AI devices
- Lab-PC+
- PC-LPM-16

- DAQCard-700
- DAQCard-1200
- DIO-24
- DIO-32F
- DIO-96

Please refer to the *SCXI Modules and Compatible DAQ Devices* section later in this chapter for information about the functionality of each DAQ device with each type of SCXI module.

SCXI Installation and Configuration

To install your SCXI system, follow the instructions in the *SCXI Hardware Installation* section in Chapter 1, *Introduction to NI-DAQ*. After you assemble your SCXI system, you must run the configuration utility to enter your SCXI configuration; NI-DAQ needs the configuration information to program your SCXI system correctly. The *SCXI Configuration in DAQCONF* and *SCXI Configuration in WDAQCONF* sections in Chapter 1 contains detailed instructions for entering your SCXI configuration using the configuration utility.

SCXI Operating Modes

The way that NI-DAQ has access to the signals from the modules depends on the operating modes of the modules. There are two basic operating modes for SCXI modules—Multiplexed and Parallel. The operating mode is a parameter that you enter in the configuration utility.

☞ **Note:** ***It is recommended that you use the Multiplexed mode.***

Multiplexed Mode for Analog Input Modules

When an analog input module operates in Multiplexed mode, all of its input channels are multiplexed to one module output. When you cable a DAQ device to a multiplexed analog input module, the DAQ device has access to that module's multiplexed output, as well as the outputs of all other multiplexed modules in the chassis via the SCXIbus. The SCXI functions route the multiplexed analog signals on the SCXIbus for you transparently. So, if you operate all modules in the chassis in Multiplexed mode, you only need to cable one of the modules directly to the DAQ device.

If you use the SCXI-1200 DAQ module in Multiplexed mode, it also has access to the multiplexed output of all analog input modules in the chassis that are operated in multiplexed mode.

If you use an MIO or AI device, Lab-PC+, or an SCXI-1200 DAQ module, you can multiplex all the analog input channels in the SCXI chassis to one onboard channel *dynamically during a timed acquisitio*n. The SCXI functions program the chassis with a module scan list that dynamically controls which module sends its output to the SCXIbus during a scan. You can specify that the modules be scanned in any order and specify an arbitrary number of channels for each module; however, the channels on each module must be scanned in consecutive, ascending order.

☞ **Note:** ***The DAQCard-700 and PC-LPM-16 support only single-channel acquisitions in Multiplexed mode.***

By default, when you cable a DAQ device to a multiplexed module, the multiplexed output of the module (and all other multiplexed modules in the chassis) appears at analog input channel 0 of the DAQ device.

You can use more than one SCXI chassis with one MIO or AI device if the modules operate in Multiplexed mode. You must use one SCXI-1350 multichassis adapter for each additional chassis (refer to your SCXI module user manuals). You should also enter a unique jumper-selected address in the configuration utility for each chassis. The multichassis adapter scheme sends the output of the module in the first chassis to analog input channel 0 of the MIO or AI device, the output of the module in the second chassis to analog input channel 1 of the MIO or AI device, and so on. When you want to acquire data from the additional chassis, you must specify the correct onboard MIO or AI device channel in the channel scan list that you pass to the SCAN functions.

You can operate the SCXI-1122 and SCXI-1100 modules only in Multiplexed mode.

Multiplexed Mode for the SCXI-1200

In Multiplexed mode, the SCXI-1200 can access the analog bus on the SCXI backplane. When you configure other analog input modules in the chassis for Multiplexed mode, the SCXI functions can multiplex their input channels and send them on the analog bus on the SCXI backplane. So, if you configure the SCXI-1200 for Multiplexed mode,

you can use it to read the multiplexed output from other SCXI analog input modules in the chassis. In addition, you can multiplex the analog input channels on the SCXI-1200 with the input channels from other analog input modules in the chassis during the same scanning operation.

You cannot use the SCXI-1200 to read channels from other analog input modules that are configured for Parallel mode.

Multiplexed Mode for Analog Output Modules

The SCXI-1124 analog output module supports only Multiplexed mode (sometimes referred to as *Serial mode* in the *SCXI-1124 User Manual*). This means that NI-DAQ sets the analog output channel states by communicating serially over the SCXIbus. Because NI-DAQ can communicate with the multiplexed modules over the SCXIbus backplane, you only need to cable one multiplexed module in each chassis directly to a DAQ device in the computer, or you can use the SCXI-1200 DAQ module to communicate with all other multiplexed modules in the chassis.

Multiplexed Mode for Digital and Relay Modules

Multiplexed mode is referred to as Serial mode in the digital and relay module hardware manuals. When you operate your digital or relay module in multiplexed (or serial) mode, NI-DAQ communicates the module channel states serially over the SCXIbus backplane. The SCXI-1162, SCXI-1162HV, SCXI-1163, and SCXI-1163R modules have jumpers that you must set correctly for the module to operate in multiplexed (or serial) mode. Because NI-DAQ can communicate with the multiplexed modules over the SCXIbus backplane, you only need to cable one multiplexed module in each chassis directly to a DAQ device in the computer, or you can use the SCXI-1200 DAQ module to communicate with all other multiplexed modules in the chassis.

Parallel Mode for Analog Input Modules

When an analog input module operates in Parallel mode, it sends each of its input channels directly to a separate analog input channel of the DAQ device cabled to the module. You cannot multiplex parallel outputs of a module on the SCXIbus; only a DAQ device that you cable directly to a module in Parallel mode has access to its input channels. In this configuration, the total number of analog input channels is limited to the number of channels available on the DAQ device. In some cases, however, you can cable more than one DAQ device to

modules in an SCXI chassis. For example, you can use two Lab-PC+ boards and cable each one to a separate SCXI-1120 module in the chassis operating in Parallel mode. You must be sure to enter the correct device numbers in the **Cabled Device** field of the configuration utility for each module you operate in Parallel mode.

By default, when a module operates in Parallel mode, the module sends its channel 0 output to analog input channel 0 of the DAQ device, the channel 1 output to analog input channel 1 of the DAQ device, and so on. You cannot use an SCXI-1200 to read channels from another analog input module in Parallel mode.

Parallel Mode for the SCXI-1200

In Parallel mode, the SCXI-1200 can read only its own analog input channels. The SCXI-1200 does not have access to the analog bus on the SCXI backplane if you configure it for Parallel mode. You should use Parallel mode if you are not using other SCXI analog input modules in the chassis in addition to the SCXI-1200.

Parallel Mode for Digital Modules

When you operate a digital module in Parallel mode, the digital lines on your DAQ device directly drive the individual digital channels on your SCXI module. You must cable a separate DAQ device directly to every module that you operate in Parallel mode. The SCXI-1162, SCXI-1162HV, SCXI-1163, and SCXI-1163R modules have jumpers that you must set correctly for the module to operate in Parallel mode. Only the DIO boards and the AT-MIO-16D can use the digital modules in Parallel mode. The MIO or AI devices, Lab-PC+, PC-LPM-16, DAQCard-700, SCXI-1200, and DAQCard-1200 cannot use the digital modules in Parallel mode.

You may wish to use Parallel mode instead of multiplexed mode for faster updating or reading of the SCXI digital channels. For the fastest performance in Parallel mode, you can use the Digital I/O functions described in Chapter 3, *Software Overview*, with the appropriate onboard port numbers instead of using the SCXI functions. Refer to the *SCXI Modules and Compatible DAQ Devices* section for information about which digital ports on each DAQ device are actually used in Parallel mode.

☞ **Note:** ***A DAQ device that is cabled to an SCXI-1162, SCXI-1162HV, SCXI-1163, and SCXI-1163R in Parallel mode cannot be the communication path in the configuration utility.***

If you are using a DIO-96 or an AT-MIO-16D, you can also operate a digital module in Parallel mode using the digital ports on the second half of the ribbon cable (pins 51–100). So, the DIO-96 can operate two digital modules in Parallel mode—one module using the first half of the ribbon cable (pins 1–50) and another module using the second half of the ribbon cable (pins 51–100). Set the operating mode in the configuration utility to **Parallel** (**secondary**) for the module that will be using the second half of the ribbon cable.

SCXI Modules and Compatible DAQ Devices

The capabilities and limitations described in this section should help you determine how your hardware components can work together in your application and help you determine the best SCXI configuration for your application. Please refer to your SCXI module and chassis user manuals and DAQ device user manuals for detailed information about the capabilities and limitations of your hardware.

The SCXI-1100

The SCXI-1100 module has 32 differential analog input channels. The input voltage range is -10 to +10 V. The SCXI-1100 has a software-selectable module gain with values of 1, 2, 5, 10, 20, 50, 100, 200, 500, 1,000, and 2,000. The gain applies to all channels on the module; use the `SCXI_Set_Gain` function to select the module gain. The gain fields in `WDAQCONF` are not used by NI-DAQ; those fields are used only by LabVIEW. The SCXI-1100 also has a software-selectable Calibration mode that you can use to determine the zero offset of the module (see the `SCXI_Calibrate_Setup` function description).

The SCXI-1300 and SCXI-1303 terminal blocks that you can use with the SCXI-1100 module each have an onboard temperature sensor that is jumper configurable to be either multiplexed with the other input channels (MTEMP configuration), or to be sent directly to a different DAQ device channel (DTEMP configuration). In the MTEMP configuration, you can select the temperature sensor in software using the `SCXI_Single_Chan_Setup` function. If you use the DTEMP configuration, the temperature sensor output appears on DAQ device channel 1. When you use the SCXI-1300, multiply the voltage you read from the temperature sensor by 100 to get degrees Celsius. The

SCXI-1303 temperature sensor is a thermistor; you can use the thermistor conversion routine described in *The Transducer Conversion Functions* section in Chapter 3, *Software Overview*, to convert the thermistor voltage to temperature.

You can cable an MIO or AI device directly to an SCXI-1100 module using the SCXI-1340 cable assembly. You must use the SCXI-1341 cable assembly for a Lab-PC+ or DAQCard-1200, and you must use the SCXI-1342 cable assembly for a PC-LPM-16 or DAQCard-700. You cannot cable a DIO board to an SCXI-1100. If you are using multiple analog input modules in Multiplexed mode, you only need to cable one of the modules in each chassis to the DAQ device, or you can control the modules using the SCXI-1200 DAQ module.

Please refer to the *SCXI-1100 User Manual* for more information on the hardware-selectable signal conditioning features on the module.

The SCXI-1100 supports only the Multiplexed operating mode; it does *not* support Parallel mode.

The SCXI-1102

The SCXI-1102 has 32 differential analog input channels and one cold-junction sensor channel (CJSTEMP) that is selectable through the `SCXI_Single_Chan_Setup` function. When you use the module with an SCXI-1300 or SCXI-1303 terminal block, the terminal block temperature sensor connects to CJSTEMP. The module can multiplex CJSTEMP with the other 32 input channels during a hardware-controlled scan. On each channel, including CJSTEMP, the SCXI-1102 has a 3-pole low-pass filter to reject 60 Hz noise. Each of the 32 differential analog input channels (but not CJSTEMP) also has an amplifier with a selectable gain of 1 or 100, selected through the `SCXI_Set_Gain` function. The amplification and filtering occur before multiplexing.

When you change the gain on a channel, the output will take several seconds to settle. The module contains a Status Register to indicate that the output is in the process of settling, and this information is available to applications through the `SCXI_Get_Status` function.

The SCXI-1102 supports only Multiplexed operating mode; it does *not* support Parallel mode.

The SCXI-1120 and the SCXI-1121

The SCXI-1120 and SCXI-1121 are 8-channel and 4-channel isolation modules, respectively. The input voltage range on both modules is ±5 V. The modules have a hardware-selectable gain on each input channel with values of 1, 2, 5, 10, 20, 50, 100, 200, 500, 1,000, and 2,000. The gain fields in `WDAQCONF` are not used by NI-DAQ; those fields are used only by LabVIEW. You use the `SCXI_Scale` function to compensate for SCXI-1120 and SCXI-1121 gains. The SCXI-1121 also has four excitation channels that you can use for voltage or current excitation.

The SCXI-1320 and SCXI-1328 terminal blocks that you can use with the SCXI-1120 and SCXI-1121 modules each have an onboard temperature sensor that is jumper-configurable to be either multiplexed with the other input channels in Multiplexed mode (MTEMP configuration), or to be sent directly to a different DAQ device channel (DTEMP configuration). In the MTEMP configuration, you can select the temperature sensor in software using the `SCXI_Single_Chan_Setup` function. If you use the DTEMP configuration, the temperature sensor output appears on DAQ device channel 15 for the SCXI-1120 and channel 4 for the SCXI-1121. Notice that the DAQ device must be in Pseudodifferential mode to read the temperature sensor in the DTEMP configuration with the SCXI-1120. When you use the SCXI-1320, multiply the voltage you read from the temperature sensor by 100 to get degrees Celsius. The SCXI-1328 temperature sensor is a thermistor; you can use the thermistor conversion routine described in *The Transducer Conversion Functions* section in Chapter 3, *Software Overview*, to convert the thermistor voltage to temperature.

You can use the SCXI-1321 terminal block only with the SCXI-1121 (Revision C and later). In addition to the features of SCXI-1320, the SCXI-1321 also has shunt resistors that you can enable using the `SCXI_Calibrate_Setup` function to check your bridge circuit. The SCXI-1327 terminal block has DIP switch configurable attenuators that can divide the input signals applied to the SCXI-1120 or SCXI-1121 by 100. You use the `SCXI_Scale` function to compensate for the attenuation when you scale your binary data to voltage.

You can cable an MIO or AI device directly to these modules using the SCXI-1340 cable assembly. You must use the SCXI-1341 cable assembly for a Lab-PC+ or DAQCard-1200, and you must use the

SCXI-1342 cable assembly for a PC-LPM-16 or DAQCard-700. You cannot cable a DIO board to an SCXI-1120 or SCXI-1121. If you are using multiple analog input modules in Multiplexed mode, you only need to cable one of the modules in each chassis to the DAQ device, or you can control the modules using the SCXI-1200 DAQ module.

Please refer to the SCXI-1120 and SCXI-1121 user manuals for information on the hardware-selectable signal conditioning features available on the modules.

The SCXI-1120 and the SCXI-1121 modules support both Multiplexed and Parallel operating modes.

The SCXI-1122

The SCXI-1122 has 16 differential analog input channels. The input voltage range is -5 V to +5 V. The SCXI-1122 has a software-selectable gain that applies to all channels on the module; use the `SCXI_Set_Gain` function to program the module gain. The gain fields in `WDAQCONF` are not used by NI-DAQ; those fields are used only by LabVIEW. This module also has a programmable low-pass filter with cut-off frequencies of 4 Hz and 4 kHz. Use the `SCXI_Configure_Filter` function to select the filter setting.

The SCXI-1122 supports Multiplexed mode only; it does not support Parallel mode.

The SCXI-1322 terminal block that can be used with the SCXI-1122 has an onboard thermistor that you can use to do cold-junction compensation for temperature readings; use the `SCXI_Single_Chan_Setup` function to select the sensor for reading. You can use the thermistor conversion routine described in *The Transducer Conversion Functions* section in Chapter 3, *Software Overview*, to convert the thermistor voltage to temperature.

You can configure the SCXI-1122 for four-wire scanning mode, which means that the module will switch the current excitation source to drive one of the channels 8 through 15 as an excitation output channel whenever the corresponding input channel 0 through 7 is selected. In this mode the module has 8 analog input channels and 8 corresponding current excitation channels. See the `SCXI_Set_Input_Mode` function description in the *NI-DAQ Function Reference Manual for PC Compatibles*.

The SCXI-1122 uses relays to switch the input channels; these relays require 10 ms to switch. As a result, you cannot use a sampling rate greater than 100 Hz in a channel-scanning operation. In addition, the relays have a finite lifetime. If you plan to take many samples from each channel and average them to eliminate noise, you should use the single-channel or software scanning applications described in the *SCXI Application Hints* section in Chapter 3, *Software Overview*. This means you should select one channel on the module, acquire many samples from that channel, then select the next channel, and so on. You should not use the channel-scanning method if you want to take many samples from each channel and average them.

The SCXI-1122 has an onboard EEPROM which contains a set of factory calibration constants for the amplifier on the module. NI-DAQ automatically reads these constants and uses them in the `SCXI_Scale` function to compensate for amplifier gain and offset errors when scaling binary data to voltage. You can also perform your own module calibration by taking readings and using the `SCXI_Cal_Constants` function to store your own calibration constants in the EEPROM.

The SCXI-1122 has two software-selectable calibration modes that you use the `SCXI_Calibrate_Setup` function to select. You can ground the module amplifier inputs so that you can read the amplifier offset. You can also switch a shunt resistor across your bridge circuit to test your circuit (refer to the *SCXI-1122 User Manual* for more information about the shunt resistor).

The SCXI-1124

The SCXI-1124 is a 6-channel analog output module capable of generating voltages between -10 and +10 V or currents between 0 and 20 mA. The SCXI-1124 has six independent 12-bit DACs. Each DAC channel has a software-selectable voltage or current output range. Use the `SCXI_AO_Write` function to set the output range and write voltages, currents, or binary values to the DACs. The SCXI-1124 is designed for single-point output operations rather than waveform generation.

The SCXI-1124 has an onboard EEPROM which contains a set of factory calibration constants for each DAC. NI-DAQ automatically loads these constants so that the `SCXI_AO_Write` function can compute the 12-bit binary pattern needed to produce your desired voltage at the output. You can also compute your own calibration

constants by writing binary values to the DACs, measuring the output voltage with a voltmeter, and using the `SCXI_Cal_Constants` function to calculate and store the constants in the module EEPROM.

The SCXI-1124 supports Multiplexed mode only. You can cable an MIO or AI device, Lab-PC+, PC-LPM-16, DAQCard-700, or DAQCard-1200 to the SCXI-1124, in which case you should set the jumpers on the module for MIO operation. You can cable a DIO board to the SCXI-1124, in which case you should set the jumpers on the module for DIO operation. If there is another module in the chassis cabled to a DAQ device in Multiplexed mode or if there is an SCXI-1200 module in the chassis, you do not need to cable the SCXI-1124 to anything; NI-DAQ will communicate with the module using the SCXIbus backplane. In this case, the MIO/DIO jumpers on the module are irrelevant. If you plan to use analog input SCXI modules in addition to the SCXI-1124, and you are not using an SCXI-1200 DAQ module, you should cable one of the analog input modules to the DAQ device.

The SCXI-1140

The SCXI-1140 is an 8-channel simultaneously sampling differential amplifier module. The input voltage range of the module is -10 to +10 V. It has a hardware-selectable gain on each input channel with values of 1, 10, 100, 200, and 500. The gain fields in `WDAQCONF` are not used by NI-DAQ; those fields are used only by LabVIEW. You use the `SCXI_Scale` function to compensate for SCXI-1140 gains. The SCXI-1140 module supports both Multiplexed mode and Parallel mode.

The SCXI-1140 will simultaneously sample all the input signals and hold those values while the DAQ device reads the desired channels one by one. When the module is holding the input channel values, it is in *Hold mode*; when it comes out of Hold mode so that it can sense the new values on the input channels, it is in *Track mode.* A control signal on the module determines when the module is in Track mode and when the module will go into Hold mode. This signal is derived either from a counter/timer output on the DAQ device, from an external source connected to a pin on the front connector of the module, or from a trigger line on the SCXIbus.

The SCXI-1140 Track/Hold setup is software-configurable for single-channel operations or for interval-scanning operations. During single-channel operations, an SCXI function call can put the module into Hold

mode before AI functions acquire the data, and put the module back into Track mode to sense new input values. During interval-scanning operations, the scan interval timer causes the module to go into Hold mode at the beginning of each scan and go back into Track mode at the end of each scan. Effectively, the input channels of the SCXI-1140 are *simultaneously sampled* at the beginning of each scan. The scan interval timer can either be a counter on the DAQ device or an external source connected to the front connector of the module.

In addition, you can synchronize multiple SCXI-1140 modules by using the SCXIbus so that all SCXI-1140 modules will go into Hold mode at the same time. If you are scanning multiple SCXI-1140 modules in Multiplexed mode along with other types of SCXI modules, the module that is cabled to the DAQ device must be an SCXI-1140 module for the Track/Hold control signals to be properly routed and synchronized.

☞ **Note:** ***Because the SCXI-1140 uses the scan interval timer of the DAQ device to control the state of the module during scanning, only DAQ devices that support interval scanning (MIO and AI devices, the Lab-PC+, DAQCard-1200, and the SCXI-1200) will support channel scanning on the SCXI-1140. The PC-LPM-16 and the DAQCard-700 do not support interval scanning, and therefore do not support timed channel scanning on the SCXI-1140 regardless of the operating mode. However, all of the DAQ devices support single-channel operations using the SCXI-1140. Please refer to*** *SCXI Application Hints* ***in Chapter 3,*** *Software Overview****, for more information on building applications with the SCXI-1140 module.***

It is important to be aware of the Track/Hold timing requirements of the SCXI-1140. For accurate data, the module must be in Track mode for at least 7 µs before going into Hold mode. During an interval-scanning operation, this means that the scan interval should be at least 7 µs longer than the total sample interval. After the module is in Hold mode, the latched data at the input channels will droop at a rate of 10 mV/s, so you must be careful to sample all the desired channels relatively quickly after putting the module into Hold mode.

The SCXI-1141

The SCXI-1141 is an 8-channel analog input module with programmable gains and filters. The input range of the module is -5 to +5 V. It has programmable gains on each channel of 1, 2, 5, 10, 20, 50, and 100; use the SCXI_Set_Gain function to program the gain on a per-channel basis. The filters have a programmable cutoff frequency

from 10 Hz to 25 kHz, and this frequency can be derived from an external clock; use the `SCXI_Configure_Filter` function to select the filter settings for the module or to enable the filters on a per-channel basis. The SCXI-1141 supports both Multiplexed and Parallel mode.

The SCXI-1141 has a software-selectable calibration mode that you can select with the `SCXI_Calibrate_Setup` function. You can ground each input of each amplifier so that you can read the amplifier offsets. The SCXI-1141 also has an onboard EEPROM that contains a set of factory gain adjustment calibration constants for each amplifier on the module. NI-DAQ automatically reads these constants and uses them in the `SCXI_Scale` function to compensate for amplifier gain errors when scaling binary data to voltage data. You can also perform your own amplifier calibration by taking readings and using the `SCXI_Cal_Constants` function to store your own calibration constants in the EEPROM.

The SCXI-1304 terminal block provides either AC or DC coupling of input signals. This terminal block also has a ground reference for floating signals.

The SCXI-1160 and the SCXI-1161

The SCXI-1160 is a 16-channel power relay module with 16 independent one-form C relays. The relays are latched—the module powers up with its relays in the position in which they were left at power down. You can set or reset each relay without affecting the other relays, or all relays can change state at the same time.

The SCXI-1161 is an 8-channel power relay module with eight independent one-form C relays. The relays are nonlatched, and the module powers up with its relays in the Normally Closed (NC) position or when the hardware reset is set on the module. You can set or reset each relay without affecting the other relays, or all the relays can change state at the same time.

The SCXI-1160 and SCXI-1161 modules only support Multiplexed (or *Serial*) mode. If you cable an MIO or AI device, Lab-PC+, DAQCard-700, DAQCard-1200, or PC-LPM-16 to one of these modules, you must set jumpers on the module to the MIO position. If you cable a DIO board to one of these modules, you must set jumpers on the module to the DIO position. If you have not cabled any device

to the module and NI-DAQ will be using the SCXIbus backplane to communicate with the module, the MIO/DIO jumpers are irrelevant.

The SCXI-1162 and SCXI-1162HV

The SCXI-1162 and SCXI-1162HV are 32-channel optically isolated digital input modules. They accept 32 input signals from external equipment and condition the signals for input to a DAQ device while maintaining optical isolation from the host computer. The SCXI-1162 accepts 0 to +5 V digital signals; the SCXI-1162HV senses AC or DC signals up to 250 V.

You can call the `SCXI_Get_State` function to read the logical states of the digital input lines on the module.

The SCXI-1162 and SCXI-1162HV modules support both Multiplexed (or *Serial*) mode and Parallel mode. You must set jumpers on the module correctly for Multiplexed or Parallel mode. If you cable an MIO or AI device, Lab-PC+, PC-LPM-16, DAQCard-700, or DAQCard-1200 to the SCXI-1162 or SCXI-1162HV, you must set jumpers on the module to the MIO position. If you cable a DIO board to the module, you must set jumpers on the module to the DIO position. If you have not cabled any device to the module and NI-DAQ will be using the SCXIbus backplane to communicate with the module, the MIO/DIO jumpers are irrelevant.

The SCXI-1163 and SCXI-1163R

The SCXI-1163 and SCXI-1163R are 32-channel optically isolated digital output modules. The SCXI-1163 makes available to external equipment up to 32 digital outputs from a DAQ device while maintaining optical isolation from the host computer and eliminating ground-loop problems. The SCXI-1163R is functionally equivalent to the SCXI-1163 but incorporates solid-state relays in place of the digital outputs. You can open or close each relay independently.

You can call the `SCXI_Set_State` function to control the digital output lines or relays of the module. You can call the `SCXI_Get_State` function to obtain the current states of the module. It is important to remember that `SCXI_Get_State` makes a hardware read only if the module is jumper-configured for and operating in Parallel mode. When operated in Serial mode, the driver retains the states of the digital output lines in memory. Consequently, a hardware write must take place before you can obtain the states on the module.

The module powers up with its digital output lines in a high state or its relays open. Calling `SCXI_Reset` also sets all the digital output lines to a high state.

The SCXI-1163 and SCXI-1163R modules support both Multiplexed (or *Serial*) mode and Parallel mode. You must set jumpers on the module correctly for Multiplexed or Parallel mode. If you cable an MIO or AI device, Lab-PC+, PC-LPM-16, DAQCard-700, or DAQCard-1200 to the SCXI-1163 or SCXI-1163R, you must set jumpers on the module to the MIO position. If you cable a DIO board to the module, you must set jumpers on the module to the DIO position. If you have not cabled any device to the module and NI-DAQ will be using the SCXIbus backplane to communicate with the module, the MIO/DIO jumpers are irrelevant.

The SCXI-1200

The SCXI-1200 is a 12-bit data acquisition and control module. NI-DAQ communicates with the SCXI-1200 through the parallel port on your PC. The SCXI-1200 is functionally similar to the Lab-PC+ plug-in DAQ device. After you configure this module in the SCXI configuration section of the configuration utility, you assign a logical device number to it. Using the device number, you can use the SCXI-1200 with almost any NI-DAQ function supported by the Lab-PC+ as if it were a plug-in device inside the PC; you can use the Analog Input functions, the Analog Output functions, the Data Acquisition functions, the Waveform Generation functions, the Digital I/O functions, and the Counter/Timer Functions, which are all described in Chapter 3, *Software Overview*. Appendix C, *NI-DAQ Function Support*, in the *NI-DAQ Function Reference Manual* shows exactly which NI-DAQ functions support the SCXI-1200.

If you configure the SCXI-1200 for Multiplexed mode, it can access the analog bus on the SCXI backplane. When you configure other analog input modules in the chassis for Multiplexed mode, the SCXI functions can multiplex their input channels and send them on the analog bus on the SCXI backplane. So, if you configure the SCXI-1200 for Multiplexed mode, you can use it to read the multiplexed output from other SCXI analog input modules in the chassis. In addition, you can multiplex the analog input channels on the SCXI-1200 with the input channels from other analog input modules in the chassis during the same scanning operation. The SCXI-1200 switches its analog input channels in descending order (the other analog input modules switch their channels in ascending order). The *SCXI Application Hints* section

in Chapter 3, *Software Overview*, explains how to use the SCXI functions in analog input applications with the SCXI-1200 in Multiplexed mode.

If you are not using other analog input modules in the chassis along with the SCXI-1200, you should configure the SCXI-1200 for Parallel mode. In Parallel mode you do not need the SCXI functions to perform analog input operations, you simply use the SCXI-1200 with the DAQ and Lab_ISCAN functions as if it were a plug-in device inside your PC. You can still use digital or relay modules or analog output modules in the chassis with the SCXI-1200 configured for Parallel mode; you need SCXI functions to operate those modules.

When you perform analog input operations with the SCXI-1200, the module sends the digitized data back to the PC using the parallel port; NI-DAQ retrieves the data from the SCXI-1200 by servicing the interrupts generated by the parallel port.

MIO and AI DAQ Devices

The MIO and AI DAQ devices support the following analog input functionality when using the SCXI analog input modules:

- Single-channel analog input using the Analog Input functions described in Chapter 3, *Software Overview*
- Single-channel data acquisition using the Data Acquisition functions described in Chapter 3, *Software Overview*
- Multiple-channel and interval scanning using the SCAN class of Data Acquisition functions described in Chapter 3, *Software Overview*

You can also use the analog output modules and digital modules in Multiplexed mode with the MIO and AI devices. If you are using analog input modules with digital or analog output modules, you must cable the MIO or AI device to an analog input module.

It is important to remember that when a DAQ device is cabled to an SCXI module, some of the DAQ device I/O connector pins and therefore some of the device resources will be reserved for SCXI use.

The following MIO and AI device resources are reserved by SCXI:

- The E Series devices use digital I/O lines 0, 1, 2, and 4 for communication with SCXI. Other digital I/O lines are available for

Digital I/O functions and can be configured for either input or output on an individual basis.

- The Am9513-based devices use digital I/O lines ADIO0 through ADIO2 to communicate with the SCXI hardware; NI-DAQ reserves those lines as output. NI-DAQ also reserves digital I/O line BDIO0 as input for SCXI communication. All of port A is reserved by NI-DAQ, but lines BDIO1 through BDIO3 are available for general use as input.
- When you use an SCXI-1140 module, the scan interval counter on the MIO or AI device controls the Track/Hold state of the module. When you set up the module for a single-channel operation, NI-DAQ reserves the scan interval counter. Refer to the `SCXI_Track_Hold_Setup` function for more information.
- The SCXI-1100 module drives analog input channel 0; if you use the SCXI terminal block with the temperature sensor in the DTEMP configuration, the SCXI-1100 also drives analog input channel 1 of the MIO or AI device.
- The SCXI-1120 module drives analog input channels 0 through 7, *even if you are operating the module in Multiplexed mode*. In addition, if the temperature sensor on the terminal block is in the DTEMP configuration, the SCXI-1120 also drives analog input channel 15. Notice that you must operate the MIO or AI device in Pseudodifferential mode to read the temperature sensor in the DTEMP configuration.
- The SCXI-1121 module drives analog input channels 0 through 3, *even if you are operating the module in Multiplexed mode*. In addition, if the temperature sensor on the terminal block is in the DTEMP configuration, the SCXI-1121 also drives analog input channel 4.
- The SCXI-1122 module drives analog input 0; if you use the SCXI terminal block with the temperature sensor in the DTEMP configuration, the SCXI-1122 also drives analog input channel 1 of the MIO or AI device.
- The SCXI-1140 module drives analog input channels 0 through 7, *even if you are operating the module in Multiplexed mode.*
- The SCXI-1141 module drives analog input channels 0 through 7, *even if you are operating the module in Multiplexed mode.*

The Lab-PC+, PC-LPM-16, DAQCard-700, SCXI-1200, and DAQCard-1200

These DAQ devices support the following analog input functionality when using the SCXI analog input modules:

- Single-channel analog input using the Analog Input functions described in Chapter 3, *Software Overview*.
- Single-channel data acquisition using the Data Acquisition functions described in Chapter 3, *Software Overview*.
- The DAQCard-700 and PC-LPM-16 support continuous channel-scanning *on the SCXI-1120 and SCXI-1121 modules only, in Parallel mode only*, using the `Lab_ISCAN` class of the Data Acquisition functions described in Chapter 3, *Software Overview*.
- The Lab-PC+, SCXI-1200, and DAQCard-1200 support continuous and interval scanning on all analog input modules in both Multiplexed and Parallel mode.

It is important to remember that when a DAQ device is cabled to an SCXI module, some of the device resources will be reserved for SCXI use. The following resources are reserved by SCXI:

- Lab-PC+, SCXI-1200, and DAQCard-1200 digital I/O lines PB4 to PB7 are used as output communication lines to SCXI. PC-LPM-16 and DAQCard-700 digital output lines DOUT4 to DOUT7 are used as output communication lines to SCXI. The entire port is reserved by NI-DAQ. If the SCXI-1200 is the only module in the chassis, these communication lines are not needed, and they are not reserved by NI-DAQ.
- Lab-PC+, SCXI-1200, and DAQCard-1200 digital I/O line PC1 is used as an input communication line to SCXI. PC-LPM-16 and DAQCard-700 digital input line DIN6 is used as an input communication line to SCXI. The remaining lines of these ports are available for input only. If the SCXI-1200 is the only module in the chassis, these communication lines are not needed, and they are not reserved by NI-DAQ.
- When you use an SCXI-1140 module, counter B1 of the Lab-PC+, SCXI-1200, or DAQCard-1200 and counter 2 of the PC-LPM-16 or DAQCard-700 control the Track/Hold state of the module. When the module is not set up for an input operation, these counters are available for general use; otherwise, they are reserved. Refer to the `SCXI_Track_Hold_Setup` function

description in the *NI-DAQ Function Reference Manual for PC Compatibles* for more information.

- The SCXI-1100 module drives analog input channel 0 of the DAQ device; if you use the SCXI-1300 terminal block with the temperature sensor in the DTEMP configuration, the SCXI-1100 also drives analog input channel 1. The SCXI-1100 cannot read the temperature sensor in DTEMP mode.
- The SCXI-1120 module drives analog input channels 0 to 7, *even if you are operating the module in Multiplexed mode.* In addition, if the temperature sensor on the terminal block is in the DTEMP configuration, the SCXI-1120 also drives analog input channel 15. Notice that the Lab-PC+ cannot read the temperature sensor on the SCXI-1320 terminal block if it is in the DTEMP configuration. The SCXI-1200 cannot read the temperature sensor in DTEMP mode.
- The SCXI-1121 module drives analog input channels 0 to 3, *even if you are operating the module in Multiplexed mode.* In addition, if the temperature sensor on the terminal block is in the DTEMP configuration, the SCXI-1121 also drives analog input channel 4. The SCXI-1200 cannot read the temperature sensor in DTEMP mode.
- The SCXI-1122 module drives analog input channel 0 of the DAQ device; if you use the SCXI-1300 terminal block with the temperature sensor in the DTEMP configuration, the SCXI-1122 also drives analog input channel 1. The SCXI-1122 cannot read the temperature sensor in DTEMP mode.
- The SCXI-1140 module drives analog input channels 0 to 7, *even if you are operating the module in Multiplexed mode*.
- The SCXI-1141 module drives analog input channels 0 to 7, *even if you are operating the module in Multiplexed mode*.

The DIO-24 and the DIO-96

The DIO-24 and DIO-96 digital I/O boards work with the digital modules and analog output modules. It is important to remember that when a digital I/O board is cabled to an SCXI module, some of the board I/O pins, and therefore some of the board resources, are reserved for SCXI use. SCXI reserves the following DIO-24 and DIO-96 resources when cabled to a digital or analog output module in Multiplexed mode:

- DIO-24 digital I/O lines PB0 to PB3 and the DIO-96 digital output lines APB0 to APB3 are the output communication lines to SCXI. The entire port is reserved by NI-DAQ.
- DIO-24 digital I/O line PA0 and DIO-96 digital I/O line APA0 are the input communication lines from SCXI. The remaining lines of these ports are available only for input.

When you cable a DIO-24 to an SCXI-1162, SCXI-1162HV, SCXI-1163, or SCXI-1163R in Parallel mode, the 24 digital lines on the DIO-24 are directly connected to channels 0 through 23 on the module; you cannot access channels 24 to 31 on the module in Parallel mode with a DIO-24.

When you cable a DIO-96 to an SCXI-1162, SCXI-1162HV, SCXI-1163, or SCXI-1163R in Parallel mode, the lines in DIO-96 ports 0 to 3 are directly connected to the channels on the module. If you configure the module for Parallel (secondary) mode and cable the second half of the ribbon cable to the module, DIO-96 ports 6 to 9 are directly connected to the digital channels on the module.

You can use the `DIG_In_Port` and `DIG_Out_Port` functions to access the SCXI-1162, SCXI-1162HV, SCXI-1163, and SCXI-1163R digital channels in Parallel mode by using the correct onboard port numbers, listed above.

You cannot cable a DIO-24 or DIO-96 to an analog input module.

The DIO-32F

The DIO-32F digital I/O board supports the digital modules and analog output modules. It is important to remember that when a digital I/O board is cabled to an SCXI module, some of the digital board I/O pins, and therefore some of the board resources, are reserved for SCXI use. SCXI reserves the following DIO-32F resources when cabled to a digital or analog output module in Multiplexed mode:

- DIO-32F digital I/O lines DIOB0 to DIOB3 are the output communication lines to the SCXI module. The entire port is reserved by NI-DAQ.
- DIO-32F digital I/O line DIOA0 is the input communication line from the SCXI module. The remaining lines of this port are available only for input.

When you cable a DIO-32F to an SCXI-1162, SCXI-1162HV, SCXI-1163, or SCXI-1163R in Parallel mode, the 32 digital lines on the DIO-32F are directly connected to the 32 channels on the module. You can use the `DIG_In_Port` and `DIG_Out_Port` functions to access the SCXI-1162, SCXI-1162HV, SCXI-1163, or SCXI-1163R channels in Parallel mode by using the correct onboard port numbers.

You cannot cable a DIO-32F to an analog input module.

Software Overview

Chapter 3

This chapter describes the classes of functions in NI-DAQ and briefly describes each function.

NI-DAQ functions are grouped according to the following classes:

- Initialization and General-Configuration functions
- Event Message functions
- Software-Calibration and Device-Specific functions
- Analog Input function group
  - One-Shot Analog Input functions
    - Single-channel Analog Input functions
    - Single-scan Analog Input (`MAI`) functions
  - Data Acquisition functions
    - High-level Data Acquisition functions
    - Low-level Data Acquisition functions
    - Low-level double-buffered Data Acquisition functions
    - `MDAQ` Data Acquisition functions
- Analog Output function group
  - One-Shot Analog Output functions
  - Waveform Generation functions
    - High-level Waveform Generation functions
    - Low-level Waveform Generation functions
- Digital I/O function group
  - Digital I/O functions
  - Group Digital I/O functions
  - Double-buffered Digital I/O functions
- Counter/Timer function group
  - Counter/Timer functions
  - Interval Counter/Timer functions

This document was created with FrameMaker 4.0.4

  - General-purpose Counter/Timer functions
- Memory Management functions
- RTSI Bus Trigger functions
- SCXI functions
- Transducer Conversion functions

The Initialization and General-Configuration Functions

Use these general functions for initializing and configuring your hardware and software:

`Align_DMA_Buffer`	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).
`Get_DAQ_Device_Info`	Retrieves parameters pertaining to the device operation.
`Get_NI_DAQ_Version`	Returns the version number of the NI-DAQ library.
`Init_DA_Brds`	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. NI-DAQ automatically calls this function; your application does not have to explicitly call it. 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.

`Master_Slave_Config`	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.
`REG_Level_Read`	Provides you with a simple method for obtaining the current state of any readable register on any supported National Instruments DAQ device. This function does not support returning the state from write-only registers.
`REG_Level_Write`	Provides you with a simple method for changing the state of any bit in any writable register on any supported National Instruments DAQ Device. NI-DAQ ensures that bits you do not wish to affect remain unchanged. This function also provides a mechanism for obtaining the current settings of write-only registers. However, you do not need to obtain the current settings of a register before writing new settings to it.
`Set_DAQ_Device_Info`	Selects parameters pertaining to the device operation.
`Timeout_Config`	Establishes a timeout limit that is used by the synchronous functions to ensure that these functions eventually return control to your application. Examples of synchronous functions are `DAQ_Op`, `DAQ_DB_Transfer`, and `WFM_from_Disk`.
`USE_*` Functions	You must call the appropriate `USE_*` function(s) at the beginning of your DOS application. It is not necessary to call these functions from a Windows,

LabWindows/CVI or Windows NT application.

The Software-Calibration and Device-Specific Functions

These software-calibration and configuration functions are each specific to only one type of device or class of devices.

`A2000_Calibrate` Calibrates the EISA-A2000 A/D gain and offset values or restores them to the original factory-set values. You can 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 a function pertaining to the EISA-A2000 is called.

`A2000_Config` 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.

`A2150_Calibrate` Performs offset calibrations on the ADCs of the specified AT-A2150.

`AO_Calibrate` 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.
`Calibrate_1200`	Calibrates the gain and offset values for the SCXI-1200, DAQPad-1200, and DAQCard-1200 ADCs and DACs. 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 store up to six sets of calibration constants. NI-DAQ automatically loads the calibration constants stored in EEPROM user area 5 the first time you call a function pertaining to the device.
`Calibrate_E_Series`	Use this function to calibrate your E Series device and to select a set of calibration constants for NI-DAQ to use.
`Configure_HW_ Analog_Trigger`	Configures the hardware analog trigger available on your E Series device.
`DSP2200_Calibrate`	Performs offset calibrations on the analog input and/or output of an AT-DSP2200.
`DSP2200_Config`	Specifies data translation and demultiplexing operations that the AT-DSP2200 can perform on analog input and output data.
`LPM16_Calibrate`	Calibrates the PC-LPM-16 converter. The function 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.

`MIO_Calibrate`	Calibrates the gain and offset values for the AT-MIO-16F-5, AT-MIO-64F-5, and AT-MIO-16X ADCs and DACs. 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 store up to six sets of calibration constants. NI-DAQ automatically loads the calibration constants stored in EEPROM user area 5 the first time you call a function pertaining to the AT-MIO-16F-5, AT-MIO-64F-5, or AT-MIO-16X.
`MIO_Config`	Turns dithering (the addition of Gaussian noise to the analog input signal) on and off. For the AT-MIO-64F-5, this function also lets you specify whether to use AMUX-64T channels or onboard channels.
`Select_Signal`	(E Series only) Chooses the source and polarity of certain signals used by the E Series devices. You typically need to use this function if you want to externally control timing, to use the RTSI bus, or to configure one of the PFI pins on the I/O connector.
`Trigger_Window_Config`	Configures the hysteresis analog trigger feature of an acquisition device. This function applies only to the AT-A2150 and the AT-DSP2200.

The Event Message Functions

NI-DAQ Event Message functions are an efficient way to monitor your background data acquisition processes.

The NI-DAQ Event Message dispatcher notifies your application when a user-specified event occurs. Using event messaging eliminates continuous polling of data acquisition processes.

`Config_Alarm_Deadband`	Specify alarm on/off condition for data acquisition event messaging.
`Config_ATrig_Event_Message`	Specify analog input trigger level and slope for data acquisition event messaging.
`Config_DAQ_Event_Message`	Specify analog input, analog output, digital input, or digital output trigger condition for event messaging.
`Get_DAQ_Event`	Get the next event message.
`Peek_DAQ_Event`	Look at the next event message.

Event Messaging Application Hints

To receive notification from the NI-DAQ data acquisition process in case of special events, you can call `Config_Alarm_Deadband`, `Config_ATrig_Event_Message`, or `Config_DAQ_Event_Message` to specify an event in which you are interested. If you are interested in more than one event, you can call any of those three functions again for each event of interest.

After you have configured all event messages, you can begin your data acquisition by calling `SCAN_Start`, `DIG_Block_In`, and so on.

When any of the events you specified occurs, NI-DAQ notifies your application.

In DOS, your application is notified through the NI-DAQ event message queue and/or a user-defined callback function. To check for event messages, you can call `Get_DAQ_Event` or `Peek_DAQ_Event`.

In Windows, notification is always made through the Windows message queue. When a user-specified event occurs, NI-DAQ puts a message into the Windows message queue. Your application receives the message when it calls the Windows GetMessage API.

After your application receives an event message, it can carry out the appropriate task, such as updating the screen or saving data to disk.

If you want to restart your data acquisition process after it completes, you do not need to call the message configuration calls again. They remain defined as long as your application does not explicitly remove them or call `Init_DA_Brds`.

If you want to add or remove a message, you must first clear your data acquisition process. Then, call one of the three event message configuration functions.

NI-DAQ Events in Visual Basic for Windows

Visual Basic Custom Controls

Unlike standard control-flow programming languages, the occurrence of events drives Visual Basic code. You interact with outside events through the properties and procedures of a control. For any given control, there is a set of procedures, called *event procedures*, that affect that control. For example, a command button named **Run** has a procedure called `Run_Click()` that is called when you click on the **Run** button. If you want something to happen when you click the **Run** button, you enter code in the `Run_Click()` procedure. When a program starts executing, Visual Basic looks for events related to controls and calls control procedures as necessary. You do not write an event loop.

There are three NI-DAQ custom controls for Visual Basic applications:

- General data acquisition (DAQ) Event (`NI-EV100.VBX`)

- Analog Trigger Event (`NI-AT100.VBX`)

- Analog Alarm Event (NI-AL100.VBX)

All of these custom controls are placed in the SYSTEM subdirectory of your Windows directory under the filenames shown above.

These three custom controls actually call the NI-DAQ Config_DAQ_Event_Message, Config_ATrig_Event_Message, and Config_Alarm_Deadband functions. Visual Basic applications cannot receive Windows messages, but by using the above NI-DAQ custom controls, your Visual Basic application can receive NI-DAQ messages.

General DAQ Event

You use the General DAQ Event custom control to configure and enable a single data acquisition event. See *The Event Message Functions* section earlier in this chapter for a complete description of NI-DAQ events. Table 3-1 lists the properties for the General DAQ Event control.

☞ **Note:** ***An n represents a generic number and is not the same value in every occurrence.***

Table 3-1. General DAQ Event Control Properties

Property	Allowed Property Values
Name	GeneralDAQEvent*n* (default)
Board	1-*n* (default)
ChanStr	See Config_DAQ_Event_Message in the *NI-DAQ Function Reference Manual*
DAQEvent	0—Acquired or generated *n* scans 1—Every *n* scans 2—Completed operation or stopped by error

Table 3-1. General DAQ Event Control Properties (Continued)

Property	Allowed Property Values
	3—Voltage out of bounds 4—Voltage within bounds 5—Analog Positive Slope Triggering 6—Analog Negative Slope Triggering 7—Digital Pattern Not Matched 8—Digital Pattern Matched 9—Counter Pulse Event
`DAQTrigVal0`	Long
`DAQTrigVal1`	Long
`TrigSkipCount`	Long
`PreTrigScans`	Long
`PostTrigScans`	Long
`Index`	
`Tag`	
`Enabled`	0—False (default) 1—True

Some General DAQ Events can be implemented only by a select group of National Instruments DAQ devices. Also, some General DAQ Events require that you set the asynchronous data acquisition or generation operation to use interrupts. For more information on the different types of General DAQ Events, refer to the descriptions for the `Config_DAQ_Event_Message` function in the *NI-DAQ Function Reference Manual*.

Each of these properties should be set as follows:

`GeneralDAQEvent`*n*.(property name) = (property value)

For example, to set the ChanStr property to Analog Input channel 0 for GeneralDAQEvent 1:

```
GeneralDAQEvent1.ChanStr = "AI0"
```

Your program flow should look like this:

1. Set the properties of the General DAQ Event control. Also, configure the acquisition or generation operations using the appropriate NI-DAQ functions.
2. Set the Enabled property of the General DAQ Event control to 1 (True).
3. Invoke the `GeneralDAQEventn.Refresh` method to actually set the DAQ Event in the NI-DAQ driver. Each subsequent invocation of `GeneralDAQEventn.Refresh` deletes the old DAQ Event and sets a new one with the current set of properties.
4. Start an asynchronous data acquisition or generation operation.
5. Whenever the selected event occurs, the `GeneralDAQEventn_Fire` procedure will be called. You can perform the necessary event processing within this procedure, such as updating a global count variable, or toggling digital I/O lines.

The `GeneralDAQEventn_Fire` procedure is prototyped as follows:

```
Sub GeneralDAQEventn_Fire (DoneFlag As Integer, Scans As Long)
```

The parameter `DoneFlag` equals 1 if the acquisition had completed when the DAQ Event fired. Otherwise it is 0. `Scans` equals the number of the scan that caused the DAQ Event to fire.

For a detailed example of how to use the General DAQ Event custom control in a Visual Basic program, please see the General DAQ Event example at the end of the *NI-DAQ Events in Visual Basic* section.

Analog Trigger Event

You use the Analog Trigger Event custom control to configure and enable an analog trigger. See *The Event Message Functions* sections earlier in this chapter for a definition of the analog trigger.

Table 3-2 lists the properties for the Analog Trigger Event control.

Table 3-2. Analog Trigger Event Control Properties

Property	Allowed Property Values
Name	GeneralDAQEvent*n* (default)
Board	1-*n* (default)
ChanStr	See Config_DAQ_Event_Message in the *NI-DAQ Function Reference Manual*
Level	Single (voltage)
WindowSize	Single (voltage)
Slope	0—Positive (default) 1—Negative
TrigSkipCount	Long
PreTrigScans	Long
PostTrigScans	Long
Index	
Tag	
Enabled	0—False (default) 1—True

The Analog Trigger Event requires that you set the asynchronous data acquisition operation to use interrupts. For more information on Analog Trigger Events, refer to the descriptions for the Config_ATrig_Event_Message function in the *NI-DAQ Function Reference Manual*.

Each of these properties should be set as follows:

AnalogTriggerEvent*n*.(property name) = (property value)

For example, to set the ChanStr property to Analog Input channel 0 for Analog Trigger Event 1:

```
AnalogTriggerEvent1.ChanStr = "AI0"
```

Your program flow should look like this:

1. Set the properties of the Analog Trigger Event control. Also, configure the acquisition or generation operations using the appropriate NI-DAQ functions.
2. Set the Enabled property of the Analog Trigger Event control to 1 (True).
3. Invoke the `AnalogTriggerEvent`*n*`.Refresh` method to actually set the Analog Trigger Event in the NI-DAQ driver. Each subsequent invocation of `AnalogTriggerEvent`*n*`.Refresh` deletes the old Analog Trigger Event and sets a new one with the current set of properties.
4. Start an asynchronous data acquisition operation.
5. Whenever the Analog Trigger conditions are met, the `AnalogTriggerEvent`*n*`_Fire` procedure will be called. You can perform the necessary event processing within this procedure, such as updating a global count variable, or toggling digital I/O lines.

The `AnalogTriggerEvent`*n*`_Fire` procedure is prototyped as follows:

```
Sub AnalogTriggerEventn_Fire (DoneFlag As Integer, Scans As Long)
```

The parameter `DoneFlag` equals 1 if the acquisition had completed when the Analog Trigger Event fired. Otherwise it is 0. `Scans` equals the number of the scan that caused the Analog Trigger Event to fire.

Analog Alarm Event

You use the Analog Alarm Event custom control to configure and enable an analog trigger. See *The Event Message Functions* sections earlier in this chapter for a definition of the analog trigger.

Table 3-3 lists the properties for the Analog Alarm Event control.

Table 3-3. Analog Alarm Event Control Properties

Property	Allowed Property Values
`Name`	GeneralDAQEvent*n* (default)
`Board`	1-*n* (default)

Table 3-3. Analog Alarm Event Control Properties (Continued)

Property	Allowed Property Values
ChanStr	See Config_DAQ_Event_Message in the *NI-DAQ Function Reference Manual*
HighAlarmLevel	Single (voltage)
LowAlarmLevel	Single (voltage)
HighDeadbandWidth	Single (voltage)
LowDeadbandWidth	Single (voltage)
Index	
Tag	
Enabled	0—False (default) 1—True

The Analog Alarm Event requires that you set the asynchronous data acquisition operation to use interrupts. For more information on Analog Alarm Events, refer to the descriptions for the Config_Alarm_Deadband function in the *NI-DAQ Function Reference Manual*.

Each of these properties should be set as follows:

AnalogAlarmEvent*n*.(property name) = (property value)

For instance, to set the ChanStr property to Analog Input channel 0 for Analog Alarm Event 1:

```
AnalogAlarmEvent1.ChanStr = "AI0"
```

Your program flow should look like this:

1. Set the properties of the Analog Alarm Event control. Also, configure the acquisition or generation operations using the appropriate NI-DAQ functions.
2. Set the Enabled property of the Analog Alarm Event control to 1 (True).
3. Invoke the AnalogAlarmEvent*n*.Refresh method to actually set the Analog Alarm Event in the NI-DAQ driver. Each

subsequent invocation of `AnalogAlarmEvent`*n*`.Refresh` deletes the old Analog Alarm Event and sets a new one with the current set of properties.

4. Start an asynchronous data acquisition operation.
5. Any one of the four following procedures can be called: `AnalogAlarm_HighAlarmOn`, `AnalogAlarm_HighAlarmOff`, `AnalogAlarm_LowAlarmOn`, or `AnalogAlarm_LowAlarmOff`. You can perform necessary event processing within this procedure, such as updating a global count variable, or toggling digital I/O lines.

The four Analog Alarm procedures are prototyped as follows:

```
Sub AnalogAlarmn_HighAlarmOn (DoneFlag As Integer, Scans As Long)
Sub AnalogAlarmn_HighAlarmOff (DoneFlag As Integer, Scans As Long)
Sub AnalogAlarmn_LowAlarmOn (DoneFlag As Integer, Scans As Long)
Sub AnalogAlarmn_LowAlarmOff (DoneFlag As Integer, Scans As Long)
```

The parameter `DoneFlag` equals 1 if the acquisition had completed when the Analog Alarm Event fired. Otherwise it is 0. `Scans` equals the number of the scan that caused the Analog Alarm Event to fire.

Using Multiple Controls

In general, a program may contain any number of General DAQ Event, Analog Trigger Event, and Analog Alarm Event controls. Just like regular Visual Basic controls, there are two ways you can place multiple controls on a Visual Basic form:

1. You can create control arrays by means of copying and pasting a control that already exists on the form. Each individual element in the control array will then be distinguished by the `Index` property, and the event procedures will have an extra parameter `Index as Integer`. The first element will have `Index` = 0, the second element will have `Index` = 1, and so on. You will only have one procedure for each type of event custom control, however, you can determine which control array element caused the event to occur by examining the `Index` property.
2. You can simply place multiple controls from the Visual Basic Tool Box on to the form. Each individual custom control of the same type will then be distinguished by the number after the `name` of the custom control, such as `GeneralDAQEvent1`,

GeneralDAQEvent2, and so on. Consequently, you will have separate procedures for each one of the custom controls, such as GeneralDAQEvent1_Fire, GeneralDAQEvent2_Fire, and so on.

General DAQ Event Example

The following steps provide an outline of how to use the General DAQ Event custom control in a Visual Basic program. A working knowledge of Visual Basic is assumed; otherwise, this example is complete. For brevity, error checking is not shown.

1. To use the GeneralDAQEvent custom control, you must first include the proper VBX file in the project window. From the **File** menu, select the **Add File** option. Look for the file NI-EV100.VBX in the \WINDOWS\SYSTEM directory. After selecting the proper file, click on the **OK** button.
2. To place the GeneralDAQEvent custom control into your form, go to the tool box window and select the GeneralDAQEvent tool, which says "DAQ EVENT" on it. Click somewhere on the form, and while holding down the mouse button, drag the mouse to place the control onto the form. You will see a little icon, which will not appear in runtime.
3. To set up a DAQ Event that notifies you after every *N* scans (DAQ Event #1), unless you decide to make *N* very large, set the device analog inputs to use interrupts by using the Set_DAQ_Device_Info function. The constants used in this function come from NIDAQCNS.INC. See the function description for Set_DAQ_Device_Info in the *NI-DAQ Function Reference Manual for PC Compatibles* and the *Programming Language Considerations* section in Chapter 1, *Using the NI-DAQ Functions*, of the *NI-DAQ Function Reference Manual for PC Compatibles* for more information. You must also configure some parameters so that the GeneralDAQEvent can occur when it needs to. In the Form_Load event routine, add the following code to what is already there:

```
er% = Set_DAQ_Device_Info(1, ND_DATA_XFER_MODE_AI, ND_INTERRUPTS)
                                          ' set AI to use INTR
GeneralDAQEvent1.Board = 1                ' assume Device 1
GeneralDAQEvent1.DAQEvent = 1             ' event every N scans
GeneralDAQEvent1.DAQTrigVal0 = 1000       ' set N=1000 scans
GeneralDAQEvent1.Enabled = True           ' If using VB1.0, set to 1
```

4. Next, you need to start some asynchronous operation. Use the NI-DAQ function `DAQ_Start`. Set up your program so it does a `DAQ_Start` on channel 0 when you click on a button you have placed on your form. To do so, add the following code in the `Command1_Click ()` subroutine as follows:

```
ReDIM buffer%(10000)
GeneralDAQEvent1.ChanStr = "AIO"
GeneralDAQEvent1.Refresh                    ‘ refresh to set params
er% = DAQ_Start(1, 0, 1, buffer%(0), 10000, 3, 10)
```

5. Next, you need to define what to do when the DAQ Event occurs. In this example, we will simply update a text box upon every 1,000 scans, and also when the whole acquisition is done. You should place a text box on your form. It will automatically be named “Text 1”. Go to the *code window*, pull down on the **Object** combo box, and select **GeneralDAQEvent1**. The only **Proc** for this control object is **Fire**. Within the subroutine, enter the following code:

```
If (DoneFlag % <> 1) Then
   Text1.Text = Str$(Scans&)+” scans have been acquired.”
Else
   Text1.Text = “Acquisition is complete!”
   er% = DAQ_Clear(1)
End If
```

6. You must make sure that you stop any ongoing acquisition when you stop the program. To do so, call the `DAQ_Clear` function before the `End` statement in the subroutine `Command2_Click ()`. You will need to place another button on your form and label it “Exit”. The subroutine should have code as follows:

```
er% = DAQ_Clear(1)
End
```

7. Run the program. Because you are not going to display the data onto a graph, it really does not matter what the data is; however, when you click on the **Click Me!** button, you should see the text box update its contents every second. After all the scans are acquired, you should see the text box display a completion message. If you run into errors, refer to the *NI-DAQ Function Reference Manual* for guidance.

8. Click on the **Exit** button to stop the program.

The Analog Input Function Group

The Analog Input function group contains two sets of functions—the One-Shot Analog Input functions, which perform single A/D conversions, and the Data Acquisition functions, which perform multiple clocked, buffered A/D functions. Within the Analog Input functions, there are two sets—single-channel Analog Input (AI) functions, which perform single A/D conversions on one channel, and multiple-channel Analog Input (MAI) functions, which perform single A/D conversions that are simultaneously sampled on a group of channels. Within the Data Acquisition functions, there are four sets—high level, low level, low-level double buffered, and MDAQ.

If you are using SCXI analog input modules (other than the SCXI-1200) you must use the SCXI functions first to program the SCXI hardware. Then you can use these functions to acquire the data using your DAQ device or SCXI-1200 module.

The One-Shot Analog Input Functions

The Single-Channel Analog Input Functions

You use the single-channel Analog Input functions for analog input on an MIO and AI device, a Lab-PC+, a PC-LPM-16, a DAQCard-500, a DAQCard-700, an SCXI-1200, a DAQPad-1200, or a DAQCard-1200:

AI_Check	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 appropriate pin; see the DAQ_Config section in Chapter 2, *Function Reference*, of the *NI-DAQ Function Reference Manual for PC Compatibles* for information on enabling external conversions.
AI_Clear	Clears the analog input circuitry and empties the FIFO memory.
AI_Configure	Informs NI-DAQ of the input mode (single-ended or differential), input

	range, and input polarity selected for the device. You must use this function if you change the jumpers affecting the analog input configuration from their factory settings. For the E Series devices, AT-MIO-16F-5, AT-MIO-64F-5, and AT-MIO-16X, which have no jumpers for analog input configuration, this function programs the device for the desired settings. 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. You also use `AI_Configure` to specify whether to drive AISENSE to onboard ground.
`AI_Mux_Config`	Configures the number of multiplexer (AMUX-64T) devices connected to the MIO and AI device and informs NI-DAQ of the presence of any AMUX-64T devices attached to the system. This function applies *only* to the MIO and AI devices.
`AI_Read`	Reads an analog input channel (initiates an A/D conversion on an analog input channel) and returns the unscaled result.
`AI_Setup`	Selects the specified analog input channel and gain setting for externally pulsed conversion operations.
`AI_VRead`	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.
`AI_VScale`	Converts the binary result from an `AI_Read` call to the actual input voltage.

Single-Channel Analog Input Application Hints

All of the NI-DAQ functions described in this section are for nonbuffered single-point analog input readings. For buffered data acquisition, consult *The Data Acquisition Functions* section later in this chapter.

Two of the AI functions are related to device configuration. If you have changed the device jumper settings from the factory-default settings or want to reprogram the E Series devices, AT-MIO-16F-5, AT-MIO-64F-5, and AT-MIO-16X, call AI_Configure at the beginning of your application to inform NI-DAQ about the changes. Furthermore, if you have connected multiplexer devices (AMUX-64T) to your MIO and AI devices, call AI_Mux_Config once at the beginning of your application to inform NI-DAQ about the multiplexer devices.

For most purposes, AI_VRead is the only function required to perform single-point analog input readings. You can use AI_Read when unscaled data is sufficient or when extra time taken by AI_VRead to scale the data is detrimental to your applications. You can use AI_VScale to convert the binary values to voltages at a later time if desired. See Figure 3-1 for the function flow typical of single-point data acquisition.

When using SCXI as a front end for analog input to an MIO and AI device, Lab-PC+, SCXI-1200, DAQPad-1200, DAQCard-1200, DAQCard-700, or PC-LPM-16, 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 obtain the unscaled data, then call the SCXI_Scale function using both the gain of the SCXI module and the gain of the DAQ device.

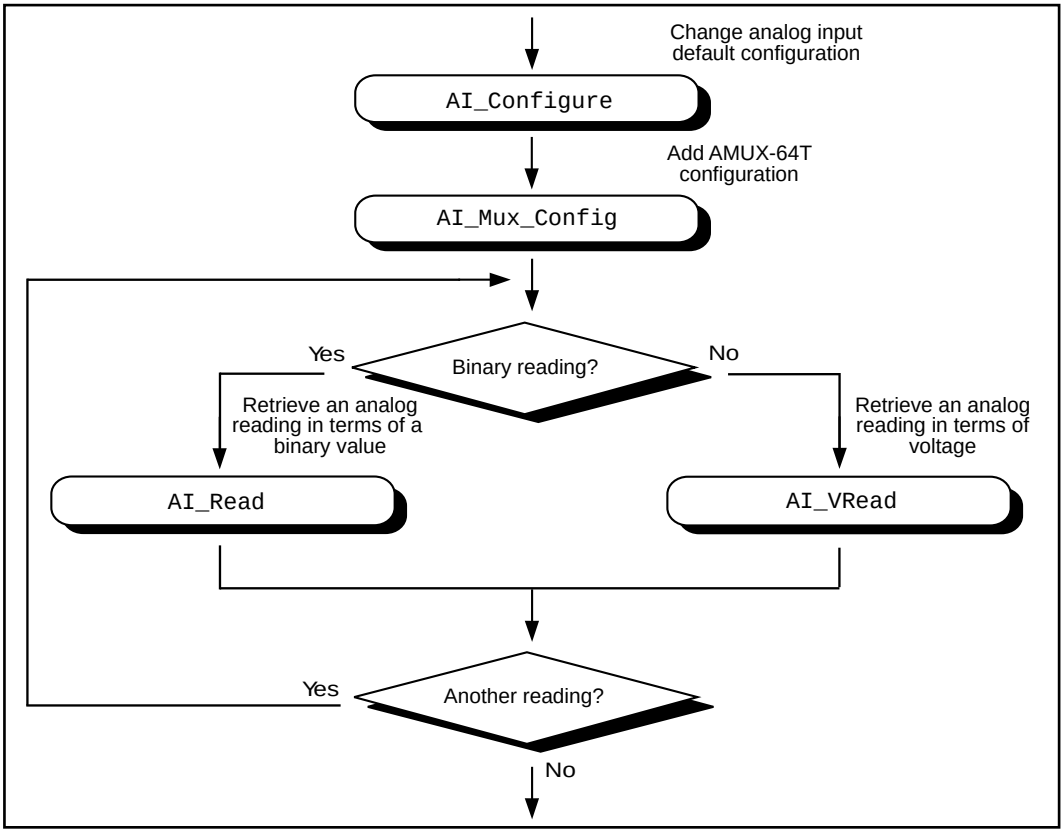


Figure 3-1. Single-Point Analog Reading with Onboard Conversion Timing

When accurate sample timing is important, you can use external conversion pulses with `AI_Clear`, `AI_Setup`, and `AI_Check` to sample your signal on the analog input channels. See Figure 3-2 for the function flow typical of single-point data acquisition using external conversion pulses. However, this method will work only if your computer is faster than the rate of conversion pulses. For high-speed data acquisition, consult *The Data Acquisition Functions* section later in this chapter for interrupt and DMA-driven data acquisition.

When using SCXI analog input modules, use the SCXI functions to set up the SCXI chassis and modules *before* using the AI functions described in Figures 3-1 and 3-2.

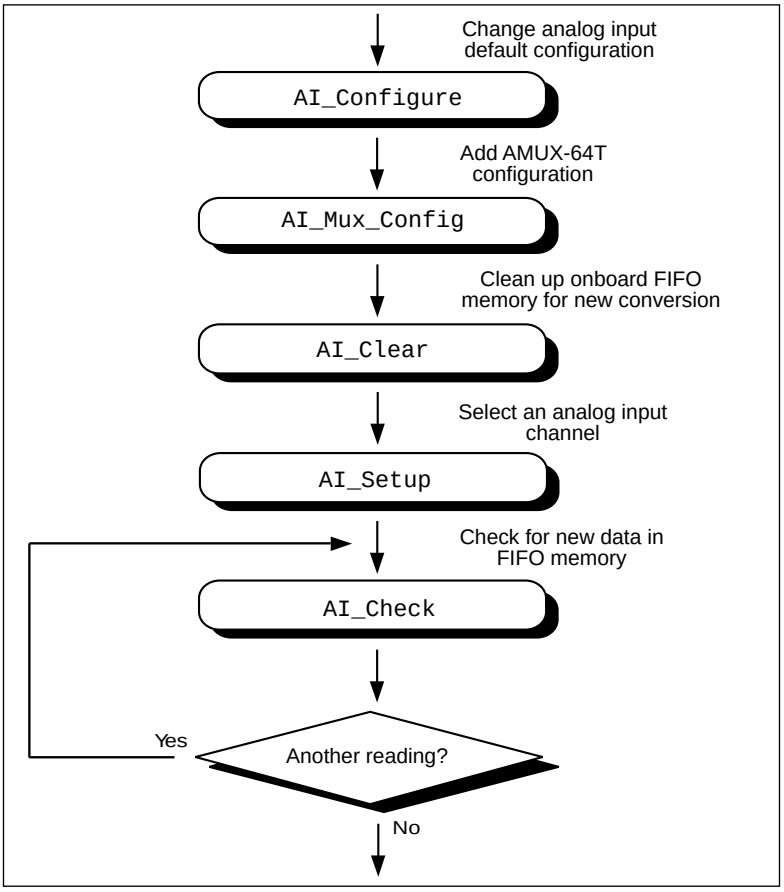


Figure 3-2. Single-Point Analog Reading with External Conversion Timing

The Single-Scan Analog Input Functions

You use the following functions for single-scan analog input operations with the EISA-A2000, AT-A2150, and AT-DSP2200 analog input boards:

`MAI_Arm` Enables/disables the board to take a sample of selected input channels whenever an external pulse on the

	SAMPCLK* input or the CLOCKI RTSI bus input is received. The function stores the data in the A/D FIFO of the board for later retrieval by MAI_Read. This function is used only by the EISA-A2000.
MAI_Clear	Clears the A/D FIFO and related analog input circuitry.
MAI_Coupling	Selects AC or DC coupling for all channels with programmable AC/DC coupling.
MAI_Read	Returns a reading for all of the selected analog input channels. If you use an external sample clock and have called MAI_Arm, this function returns samples generated by previous sample clock pulses; otherwise, this function clears the A/D FIFO, generates an A/D conversion pulse, and returns the samples produced by this pulse.
MAI_Scale	Given an array of acquired data, MAI_Scale converts the values in the array to the actual voltage values measured.
MAI_Setup	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.

The MAI functions perform single A/D conversions simultaneously sampled on a group of channels.

Single-Scan Analog Input Application Hints

For most operations, MAI_Read is the only function required to perform a single scan of all the analog input channels. Use

`MAI_Scale` to subsequently convert the binary values to voltage values. Use `MAI_Setup` to change the analog input channels monitored. Use `MAI_Coupling` to select AC or DC coupling on the analog input channels.

The default settings for analog input with the EISA-A2000 are as follows:

- AC coupling on all input channels
- Four analog input channels (channels 0 through 3) selected
- Internal, onboard sample clock used

The default settings for analog input with the AT-A2150 are as follows:

- DC coupling on all input channels
- Four analog input channels (channels 0 through 3) selected
- Internal, onboard sample clock used

The default settings for analog input with the AT-DSP2200 are as follows:

- DC coupling on all input channels
- Two analog input channels (channels 0 and 1) selected
- Internal, onboard sample clock used

Figure 3-3 shows the function flow typical of single-scan analog input readings.

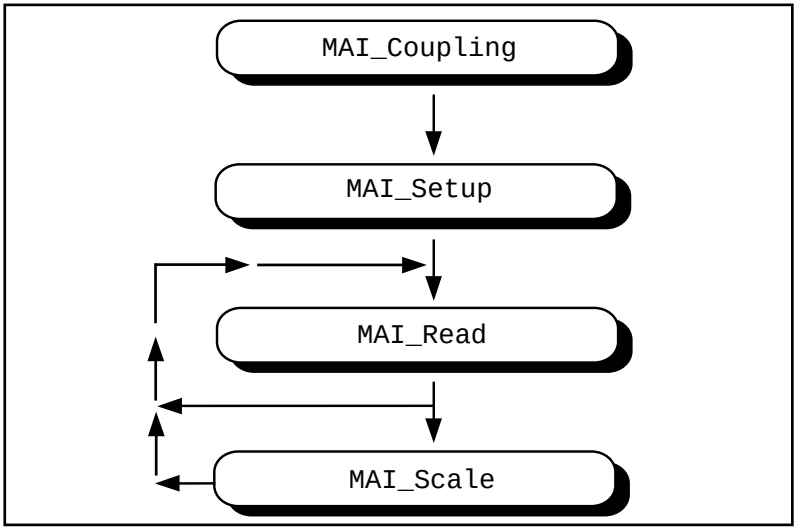


Figure 3-3. Function Flowchart for Single-Scan Analog Input Readings

After startup or a board reset, the MAI_Read function returns a reading from all available analog input channels. The MAI_Setup step is necessary only if the number of channels needed is different than the default. The MAI_Scale step is optional, although many applications perform this step for every MAI_Read done to convert the reading to the actual voltage values measured.

Buffered Analog Input

You implement buffered, multiple-channel analog input with the MDAQ functions described later in this chapter. You can also use MAI_Coupling, MAI_Setup, MAI_Arm, and MAI_Scale with the MDAQ functions.

Externally Clocked Analog Input with the EISA-A2000

Use MAI_Arm and MAI_Clear only for externally clocked sampling. Use A2000_Config to select an external sample clock. If you use the CLOCKI input of the RTSI bus, call RTSI_Conn at this time to make the connection. Otherwise, use the SAMPCLK* input of the I/O connector. Use MAI_Arm to enable the EISA-A2000 to sample its inputs and save the readings in the A/D FIFO whenever it receives a sample clock edge. Call MAI_Read to retrieve the readings. MAI_Read returns the earliest sample in the A/D FIFO for the

channels selected, or an error if no readings are present. Use MAI_Clear at any time to clear the A/D FIFO and any error conditions. You can use MAI_Arm again to disable externally clocked analog input. In Figure 3-4, the pulses are received at the SAMPCLK* input.

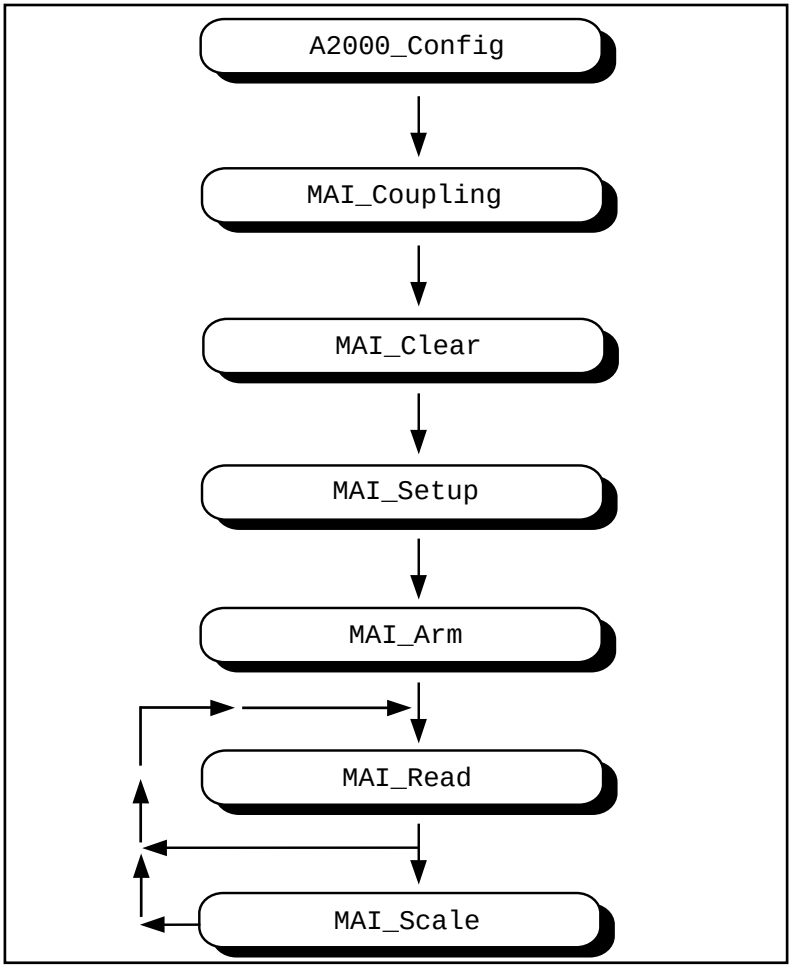


Figure 3-4. Flowchart for an Externally Clocked Analog Input

MAI_Coupling, MAI_Setup, and MAI_Scale are optional calls.

The Data Acquisition Functions

Use the SCAN, DAQ, Lab, and DAQ_DB functions with the following analog input devices:

- DAQCard-500
- DAQCard-700
- DAQCard-1200
- DAQPad-1200
- Lab-PC+
- MIO and AI devices
- PC-LPM-16
- SCXI-1200

The High-Level Data Acquisition Functions

These high-level data acquisition functions are synchronous calls that acquire data and return when data acquisition is complete.

DAQ_Op	Performs a synchronous, single-channel data acquisition operation. DAQ_Op does not return until NI-DAQ has acquired all the data or an acquisition error has occurred.
DAQ_to_Disk	Performs a synchronous, single-channel data acquisition operation and saves the acquired data in a disk file. DAQ_to_Disk does not return until NI-DAQ has acquired all the data and saved or an acquisition error has occurred.
Lab_ISCAN_Op	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, DAQCard-700, DAQCard-1200, DAQPad-1200, Lab-PC+, PC-LPM-16, and SCXI-1200 only).

`Lab_ISCAN_to_Disk`	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 all the data and saved or an acquisition error has occurred (DAQCard-500, DAQCard-700, DAQCard-1200, DAQPad-1200, Lab-PC+, PC-LPM-16, and SCXI-1200 only).
`SCAN_Op`	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).
`SCAN_to_Disk`	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 NI-DAQ has acquired all the data and saved or an acquisition error has occurred (MIO and AI devices only).

The Low-Level Data Acquisition Functions

These functions are low-level primitives used for setting up, starting, and monitoring asynchronous data acquisition operations.

`DAQ_Check`	Checks if the current data acquisition operation is complete and returns the status and the number of samples acquired to that point.
`DAQ_Clear`	Cancels the current data acquisition operation (both single-channel and multiple-channel scanned) and reinitializes the data acquisition circuitry.

DAQ_Config	Stores configuration information for subsequent data acquisition operations.
DAQ_Monitor	Returns data from an asynchronous data acquisition in progress. During a multiple-channel acquisition, you can call DAQ_Monitor to retrieve data from a single channel or from all channels being scanned. Using the Oldest/Newest mode, you can specify whether DAQ_Monitor returns sequential (oldest) blocks of data, or the most recently acquired (newest) blocks of data.
DAQ_Rate	Converts a data acquisition rate into the timebase and sample-interval values needed to produce the desired rate. This function also supports the EISA-A2000.
DAQ_Start	Initiates an asynchronous, single-channel data acquisition operation and stores its input in an array.
DAQ_StopTrigger_Config	Enables the pretrigger mode of data acquisition and indicates the number of data points to acquire after you apply the stop trigger pulse at the appropriate pin.
DAQ_VScale	Converts the values of an array of acquired binary data and the gain setting for that data to actual input voltages measured.
Lab_ISCAN_Check	Checks if the current scan data acquisition operation 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, DAQCard-700, DAQCard-1200, DAQPad-1200, Lab-PC+, PC-LPM-16, and SCXI-1200 only).
`Lab_ISCAN_Start`	Initiates a multiple-channel scanned data acquisition operation and stores its input in an array (DAQCard-500, DAQCard-700, DAQCard-1200, DAQPad-1200, Lab-PC+, PC-LPM-16, and SCXI-1200 only).
`SCAN_Demux`	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.
`SCAN_Sequence_Demux`	Rearranges the data produced by a multirate acquisition so that all the data from each channel is stored in adjacent elements of your buffer.
`SCAN_Sequence_ Retrieve`	Returns the scan sequence created by NI-DAQ as a result of a previous call to `SCAN_Sequence_Setup`.
`SCAN_Sequence_Setup`	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.
`SCAN_Setup`	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).

`SCAN_Start` — 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).

The Low-Level Double-Buffered Data Acquisition Functions

These functions are low-level primitives used for setting up and monitoring asynchronous double-buffered data acquisition operations:

`DAQ_DB_Config` — Enables or disables double-buffered data acquisition operations.

`DAQ_DB_HalfReady` — 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.

`DAQ_DB_StrTransfer` — 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 streamed-to-disk as it is acquired. You can then write the string to a disk file using the BASIC `PUT` statement.

`DAQ_DB_Transfer`	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.

Data Acquisition Application Hints

Counter/Timer Signals on the Lab-PC+, SCXI-1200, DAQPad-1200, and DAQCard-1200

For the Lab-PC+, SCXI-1200, DAQCard-1200, and DAQPad-1200, counter A2 produces the total sample interval for data acquisition timing. However, if the total sample interval is greater than 65,535 µs, counter B0 generates the clock for a slower timebase, which counter A2 uses for the total sample interval. Thus, the `ICTR_Setup` and `ICTR_Reset` functions cannot use counter B0 for the duration of the data acquisition operation.

In addition, the Waveform Generation functions cannot use counter B0 if the total update interval for waveform generation is also greater than 65,535 µs and counter B0 must produce a timebase for waveform generation that is different from the timebase counter B0 produced for data acquisition. If waveform generation is not in progress, counter B0 is available for data acquisition if you have made no `ICTR_Setup` call on counter B0 since startup or you have made an `ICTR_Reset` call on counter B0. If waveform generation is in progress and is using counter B0 to obtain the timebase required to produce the total update interval, counter B0 is available for data acquisition only if this timebase is the same as that required by the Data Acquisition functions to produce the total sample interval. In this case, counter B0 provides the same timebase for data acquisition and waveform generation.

Counter/Timer Signals on the DAQCard-500, DAQCard-700, and PC-LPM-16

For the DAQCard-500, DAQCard-700, and PC-LPM-16, counter 0 produces the sample interval for data acquisition timing. If data acquisition is not in progress, you can call the `ICTR` functions to use

counter 0 as a general-purpose counter. Because the CLOCK0 input is connected to a 1 MHz oscillator, the timebase for counter 0 is fixed.

External Multiplexer Support (AMUX-64T)

You can expand the number of analog input signals that the MIO and AI devices can measure with an external multiplexer device (AMUX-64T). See *The AMUX-64T External Multiplexer Device* of Chapter 2, *Hardware Overview*, for more information on using the AMUX-64T with your MIO and AI device. See the *AMUX-64T User Manual* for more information on the external multiplexer device.

Basic Building Blocks

Most of the buffered data acquisition applications are made up of four building blocks, as shown in Figure 3-5. However, depending on the specific devices and applications you have, the NI-DAQ functions that make up each building block vary. Typical applications may include these NI-DAQ functions in each of their four building blocks.

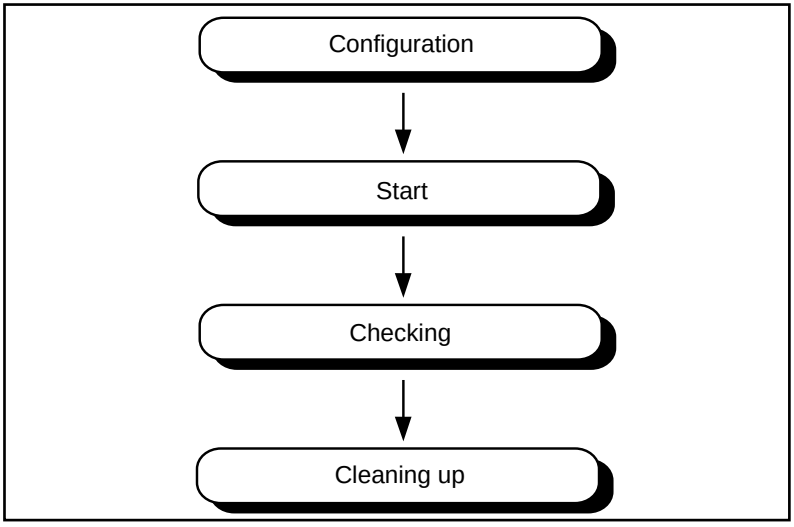


Figure 3-5. Buffered Data Acquisition Basic Building Blocks

When using SCXI analog input modules, use the SCXI functions to set up the SCXI chassis and modules before using the `AI`, `DAQ`, `SCAN`, and `Lab_ISCAN` functions shown in the following flowcharts.

Building Block 1: Configuration

Five configuration functions are available for creating the first building block, as shown in Figure 3-6. However, you do not have to call all five functions every time you start a data acquisition.

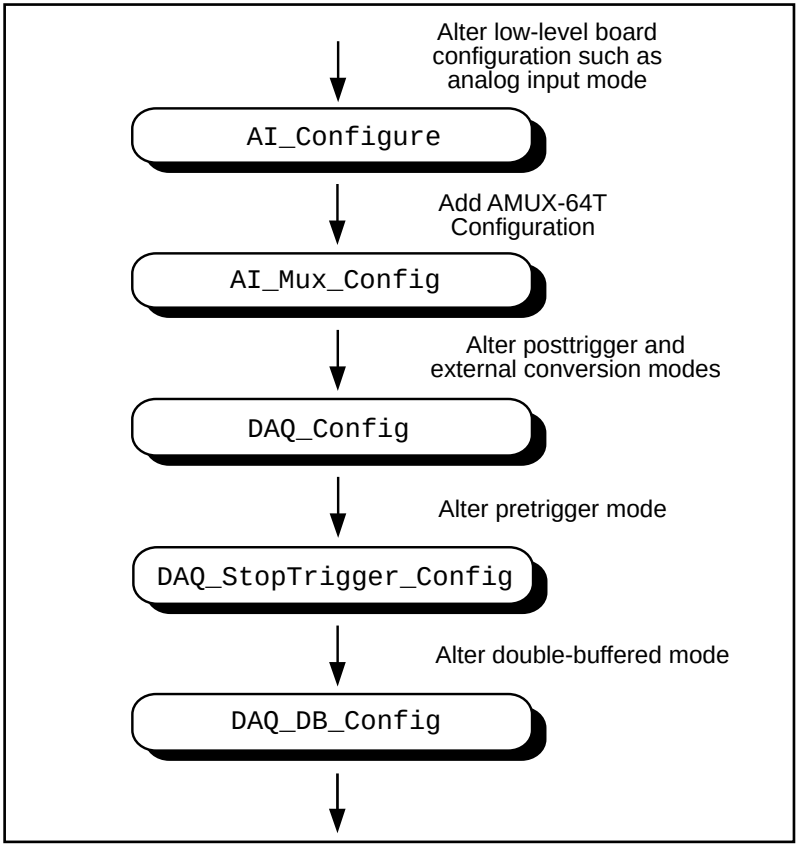


Figure 3-6. Buffered Data Acquisition Application Building Block 1, Configuration

NI-DAQ always records the device configurations and the default configurations. (See the `Init_DA_Brds` description in the *NI-DAQ Function Reference Manual* for device default configurations.) Therefore, if you are satisfied with the default or the current configurations of your devices, your configuration building block will be empty, and you can go on to the next building block, Start.

Building Block 2: Start

NI-DAQ has high-level and low-level start functions. The high-level start functions are as follows:

- `DAQ_Op`
- `SCAN_Op` (MIO and AI devices only)
- `Lab_ISCAN_Op` (DAQCard-500, DAQCard-700, DAQCard-1200, DAQPad-1200, Lab-PC+, PC-LPM-16, and SCXI-1200 only)
- `DAQ_to_Disk`
- `SCAN_to_Disk` (MIO and AI devices only)
- `Lab_ISCAN_to_Disk` (DAQCard-500, DAQCard-700, DAQCard-1200, DAQPad-1200, Lab-PC+, PC-LPM-16, and SCXI-1200 only)

A high-level start call initiates data acquisition but does not return to the function caller until the data acquisition is complete. For that reason, you do not need the next building block, Checking, when you use high-level start functions.

The major advantage of the high-level start functions is that they are simple. A single call can give you a buffer full or a disk full of data. However, if your application is acquiring data at a very slow rate or is acquiring a lot of data, the high-level start functions may tie up the computer for a significant amount of time. Therefore, NI-DAQ has some low-level (or asynchronous) start functions that initiate data acquisition and return to the function caller immediately.

Asynchronous start functions include DAQ_Start, SCAN_Start, and Lab_ISCAN_Start functions. Figures 3-7 and 3-8 show how the start calls make up building block 2.

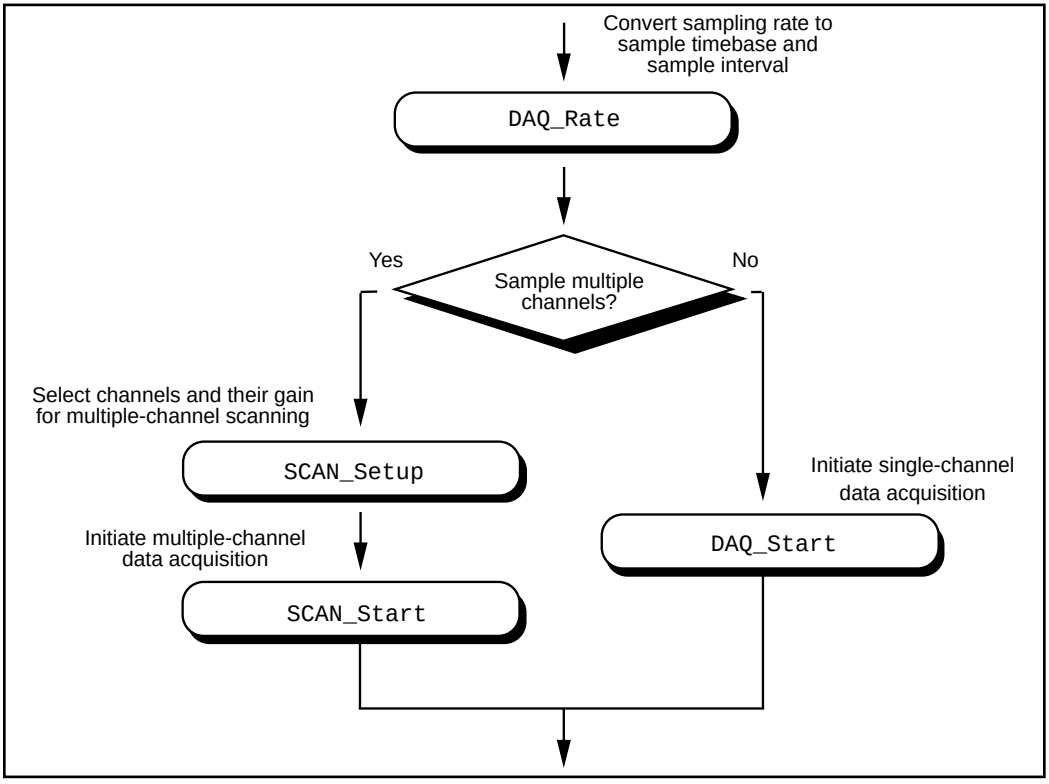


Figure 3-7. Buffered Data Acquisition Application Building Block 2, Start, for the MIO and AI Devices

If your device supports multirate scanning (scanning different channels at different rates), you can use SCAN_Sequence_Setup instead of SCAN_Setup in building block 2.

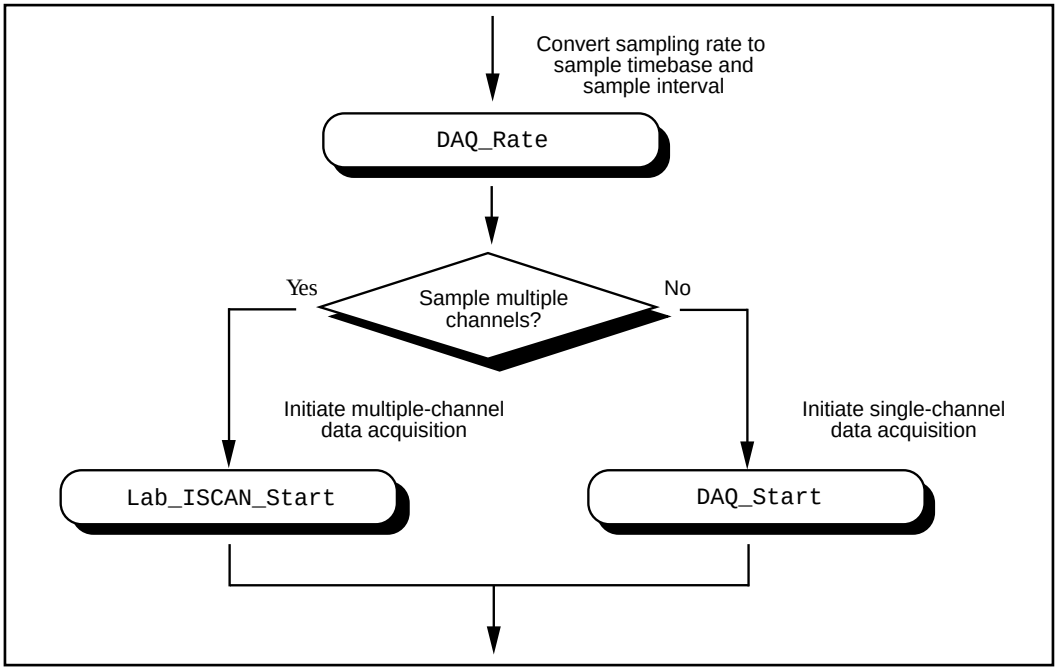


Figure 3-8. Buffered Data Acquisition Application Building Block 2, Start, for the Lab-PC+, DAQCard-500, DAQCard-700, PC-LPM-16, SCXI-1200, DAQPad-1200, and DAQCard-1200

When you have the asynchronous start calls in your building block 2, the next building block, Checking, will be very useful for finding out the status of the ongoing DAQ process.

Building Block 3: Checking

DAQ_Check and Lab_ISCAN_Check shown in Figures 3-9 and 3-10 are simple and quick ways to check the ongoing DAQ process.

This call is often put in a while loop so that the application can periodically monitor the DAQ process.

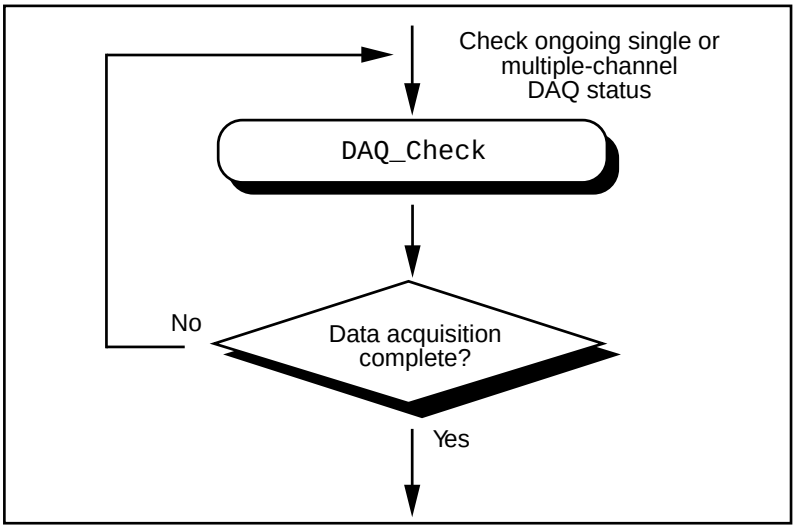


Figure 3-9. Buffered DAQ Application Building Block 3, Checking, for the MIO and AI Devices

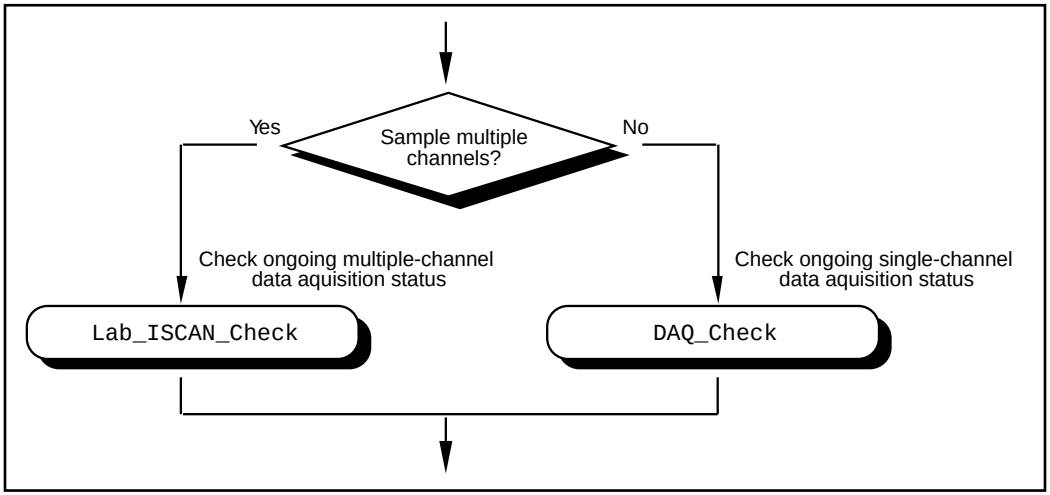


Figure 3-10. Buffered Data Acquisition Application Building Block 3, Checking, for the Lab-PC+, DAQCard-500, DAQCard-700, PC-LPM-16, SCXI-1200, DAQPad-1200, and DAQCard-1200

However, if the information provided by DAQ_Check does not satisfy your needs, DAQ_Monitor or the double-buffered functions may be a better choice. With DAQ_Monitor, you can not only monitor the DAQ process but also retrieve a portion of the acquired data. With the double-buffered functions, you can retrieve half of the data buffer at a time. Double-buffered functions are very useful when your application has a real-time strip chart displaying the incoming data.

Building Block 4: Cleaning Up

The purpose of this building block is to stop the data acquisition and free any system resources (such as DMA channels) used for the data acquisition. DAQ_Clear is the only function needed for this building block and is automatically called by the check functions described in the previous building block when the data acquisition is complete. Therefore, you can eliminate this last building block if your application continuously calls the previously described check functions until the data acquisition is complete.

☞ **Note:** DAQ_Clear ***does not alter the device configurations made by building block 1.***

Double-Buffered Data Acquisition

The double-buffered (DAQ_DB) Data Acquisition functions return data from an ongoing data acquisition without interrupting the acquisition. These functions use a double, or circular, buffering scheme that permits half buffers of data to be retrieved and processed as the data becomes available. By using a circular buffer, you can collect an unlimited amount of data without needing an unlimited amount of memory. Double-buffered data acquisition is useful for applications such as streaming data to disk and real-time data display.

Initiating double-buffered data acquisition requires some simple changes to the first and third basic building blocks, Configuration and Checking, respectively.

In building block 1, turn on double-buffered mode data acquisition through the DAQ_DB_Config call. Notice that after double-buffered mode is enabled, all subsequent data acquisitions will be in double-buffered mode.

In building block 3, different checking functions are needed. Figure 3-11 shows a simple way to monitor the data acquisition in progress and to retrieve data when they are available.

For further details on double-buffered data acquisition, consult Chapter 6, *NI-DAQ Double Buffering*.

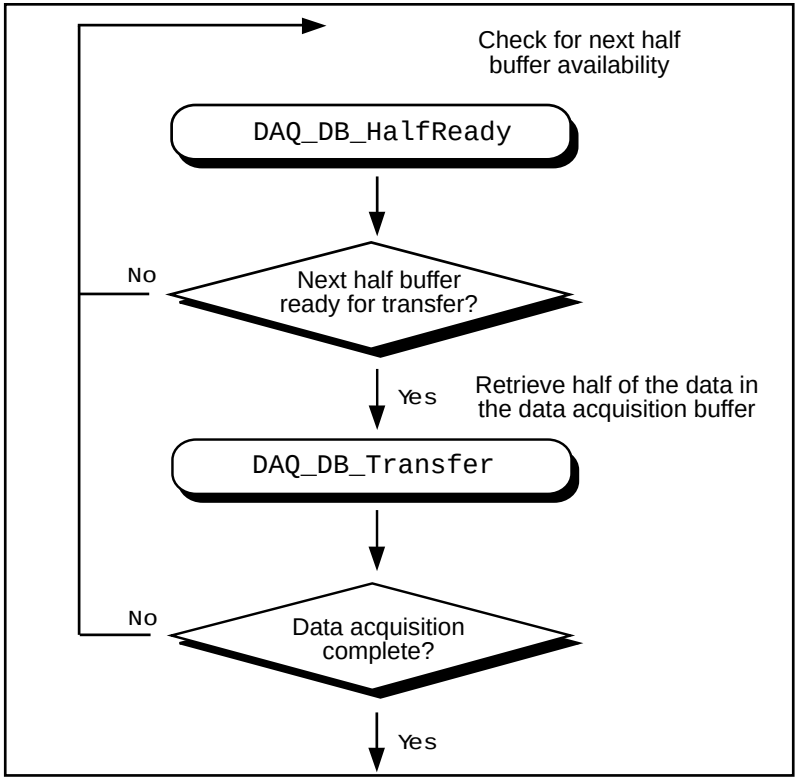


Figure 3-11. Double-Buffered Data Acquisition Application Building Block 3, Checking

Multirate Scanning

You can use multirate scanning to scan multiple channels at different scan rates and acquire the minimum amount of data necessary for your application. This is particularly useful if you are scanning very fast and want to write your data to disk, or if you are acquiring large amounts of data and want to keep your buffer size to a minimum.

Multirate scanning is a hardware-dependent feature that is implemented for the AT-MIO-16F-5, AT-MIO-64F-5, AT-MIO-16X, and all E Series devices.

Multirate scanning works by scanning each channel at a rate that is a fraction of the specified scan rate. For example, if you want to scan four channels at 6,000, 4,000, 3,000, and 1,000 scans per second, you specify a scan rate of 12,000 scans per second and a scan rate divisor vector of 2, 3, 4, and 12.

NI-DAQ includes three functions for multirate scanning:

- `SCAN_Sequence_Setup`
- `SCAN_Sequence_Retrieve`
- `SCAN_Sequence_Demux`

You can use `SCAN_Sequence_Setup` to identify the channels to scan, their gains, and their scan rate divisors. After the data is acquired, use `SCAN_Sequence_Retrieve` and `SCAN_Sequence_Demux` to arrange the data into a more convenient format.

Figure 3-12 shows how to use the multirate scanning functions in conjunction with other NI-DAQ functions.

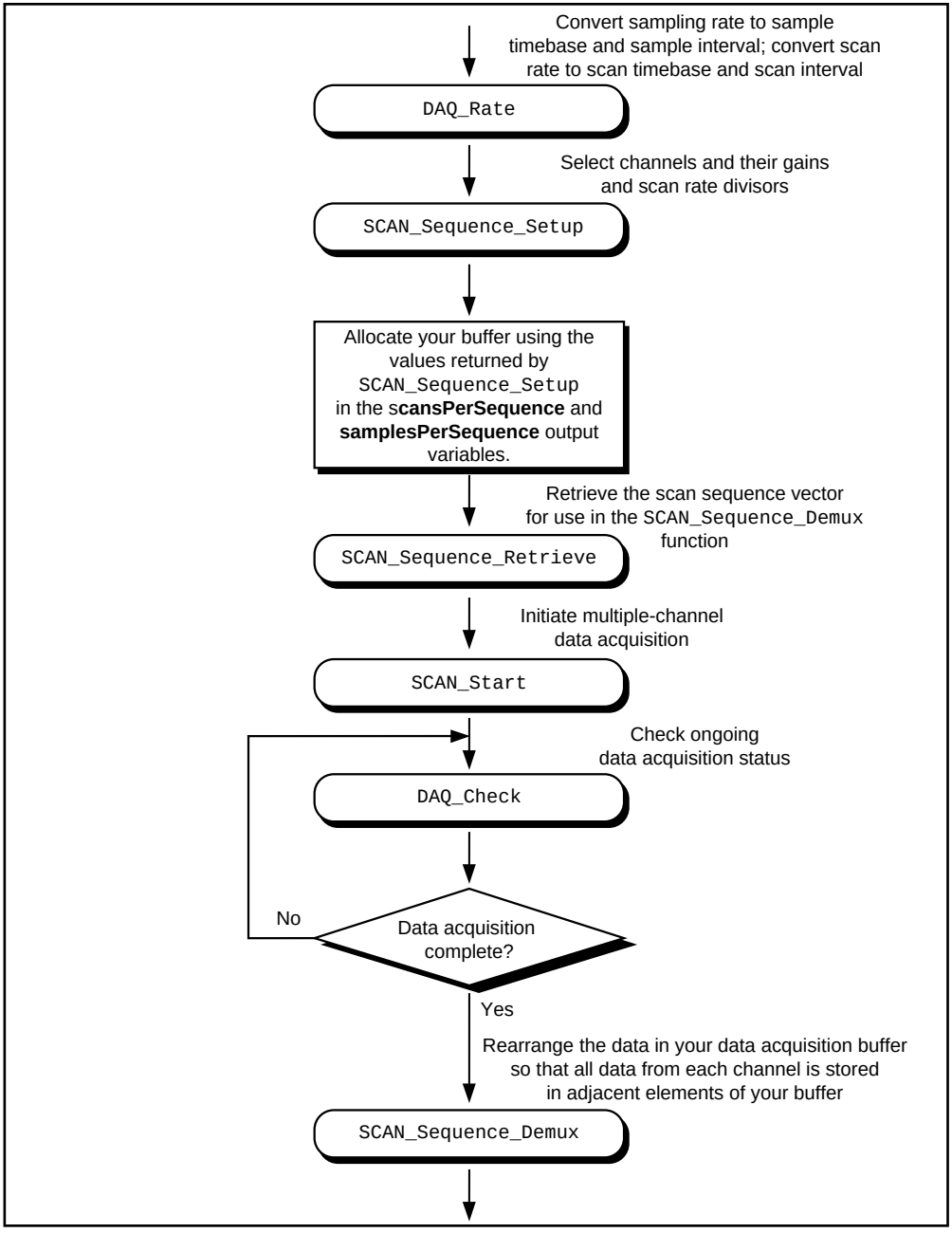


Figure 3-12. Multirate Scanning

The MDAQ Functions

Use the following functions for multiple-channel buffered data acquisition operations on the EISA-A2000, AT-A2150, and AT-DSP2200:

`MDAQ_Check`	Reports the current status of the data acquisition—whether the data acquisition is complete—and the number of acquired frames and scans.
`MDAQ_Clear`	Stops data acquisition, and releases internal resources so that your application can successfully terminate.
`MDAQ_Get`	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_Get` can retrieve data from anywhere in the acquisition buffer.
`MDAQ_ScanRate`	Selects the data acquisition scan rate—the rate at which all selected input channels are sampled.
`MDAQ_Setup`	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.
`MDAQ_Start`	Starts a multiple-channel data acquisition operation.
`MDAQ_Stop`	Stops the data acquisition but leaves all settings in effect.
`MDAQ_StrGet`	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.

MDAQ_Trig_Delay	Selects the time to delay after a trigger is received before acquiring data (Posttrigger mode only).
MDAQ_Trig_Select	Selects the trigger source and configures the analog and digital trigger conditions.

MDAQ Application Hints

The MDAQ functions perform both single-channel and multiple-channel data acquisition operations. The following terminology is used to describe the MDAQ functions:

- A *frame* is a set of samples acquired from all selected channels with each trigger. The number of samples per frame is equal to (*pretrig_scans* + *posttrig_scans*) ∗ number of channels selected.
- A *scan* is one sample from each of the selected analog input channels. A scan can consist of 1, 2, or 4 samples.
- The *scan_interval* is the time between the initiation of consecutive *scans*. The *scan_interval* is equivalent to the interval between samples on a given channel.
- The *pretrig_scans* is the number of *scans* to acquire before the trigger.
- The *posttrig_scans* is the number of *scans* to acquire after the trigger.

For both single read analog input and data acquisition, MAI_Setup selects the analog input channels to be sampled and MAI_Coupling selects AC or DC coupling for all inputs.

Frame-Oriented and Scan-Oriented Data Acquisition

You can perform data acquisition with the EISA-A2000, AT-A2150, and AT-DSP2200 in two modes—frame-oriented data acquisition or scan-oriented data acquisition. The **scansOrFrames** parameter in MDAQ_Setup determines which mode is used.

In frame-oriented data acquisition mode, the device acquires multiple frames. The device acquires a frame each time the device receives a trigger. Each frame can contain both pretrigger and posttrigger data. All frames are the same size and use the same trigger modes, number of channels, and acquisition rates. MDAQ_Setup configures the frame size and the number of pretrigger and posttrigger scans. MAI_Setup

selects the analog input channels and MDAQ_ScanRate configures the acquisition rate. Using the MDAQ_Start function, you can either specify a number of frames to be acquired, or the device can acquire an unlimited number of frames until you stop acquisition (MDAQ_Stop).

The scan-oriented data acquisition mode is a posttrigger, single-frame data acquisition case. After the device receives a trigger, the device acquires either a specified number of scans before the acquisition is automatically stopped, or an unlimited number of scans until you stop the acquisition. If you specify a finite number of scans, this number can be larger than the frame size. The scan-oriented mode is useful for streaming data to disk using MDAQ_Get.

Configuring the Trigger Conditions

The MDAQ_Setup, MDAQ_Trig_Select, and MDAQ_Trig_Delay functions configure the acquisition triggering conditions. The EISA-A2000, AT-A2150, and AT-DSP2200 have both digital and analog triggering capability. The selection and enabling of the triggers is performed in MDAQ_Trig_Select, and this function can arm both the analog and the digital trigger, as well as disable both triggers (thereby configuring a software-triggered acquisition). When the acquisition is in posttrigger mode, the MDAQ_Trig_Delay function can set a delay between the trigger and the conversion of the first scan.

Stopping Data Acquisition

MDAQ_Stop or MDAQ_Clear can halt data acquisition in progress. This is necessary when continuously scanning or when gathering an unlimited number of frames. MDAQ_Stop does not affect any data still present in the A/D FIFO (only the EISA-A2000 and the AT-A2150 have FIFOs), does not prepare the system for the termination of your application, and maintains the full functionality of the MDAQ_Get call. MDAQ_Stop is useful to simply stop the acquisition 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_Clear prepares the system for terminating the application. In addition to halting the acquisition, MDAQ_Clear clears the A/D FIFO and releases certain internal buffers and system resources. One result is that the MDAQ_Get call is unable to *unwrap* pretrigger data. You should always call MDAQ_Clear before your application terminates.

Using Onboard AT-DSP2200 Memory

The AT-DSP2200 can store incoming analog input in its own onboard memory. To do this, you must allocate a buffer of onboard memory using the `NI_DAQ_Mem_Alloc` function or the DSP memory allocation call in the NI-DSP library. You can use the DSP memory handle as the **acqBuffer** in `MDAQ_Setup` or the **getBuffer** in `MDAQ_Get`. Storing analog input data on the AT-DSP2200 board makes sense when an NI-DSP function is to operate on the DSP board. The AT-DSP2200 can also translate analog input data into floating-point format and sort the data by channel as it is written to onboard DSP memory.

Typical MDAQ Function Usage

A typical function order needed to start an acquisition is as follows:

1. `MAI_Coupling` to select coupling on input channels
2. `MAI_Setup` to select the number of channels to monitor
3. `MDAQ_Setup` to select frame size and the number of pretrigger and posttrigger scans
4. `MDAQ_ScanRate` to select acquisition rate
5. `MDAQ_Trig_Select` to select trigger type and conditions
6. `MDAQ_Trig_Delay` to select posttrigger delay (if triggering is enabled and the posttrigger mode is used)
7. `MDAQ_Start` to select the number of frames to acquire and to start acquisition

While the device is acquiring data, you can use the following functions:

- `MDAQ_Check` to monitor the status of the acquisition
- `MDAQ_Get` to fetch data from anywhere in the acquisition buffer
- `MDAQ_Stop` to stop the data acquisition, after which you can use `MDAQ_Get` to fetch the data
- `MDAQ_Clear` to prepare the application for termination

The following flowchart figures illustrate the ways to use the `MDAQ` functions. Two examples of performing a data acquisition are shown, followed by a more detailed look at the possible combinations of functions used initially, during, and after the completion of an acquisition.

The functions in Figure 3-13 are the minimal set of functions necessary to acquire a given number of untriggered frames (that is, with no triggering enabled), sampling all four channels at 250,000 scans/s for an EISA-A2000, 32,000 scans/s for an AT-A2150C and AT-DSP2200, and 16,000 scans/s for an AT-A2150S.

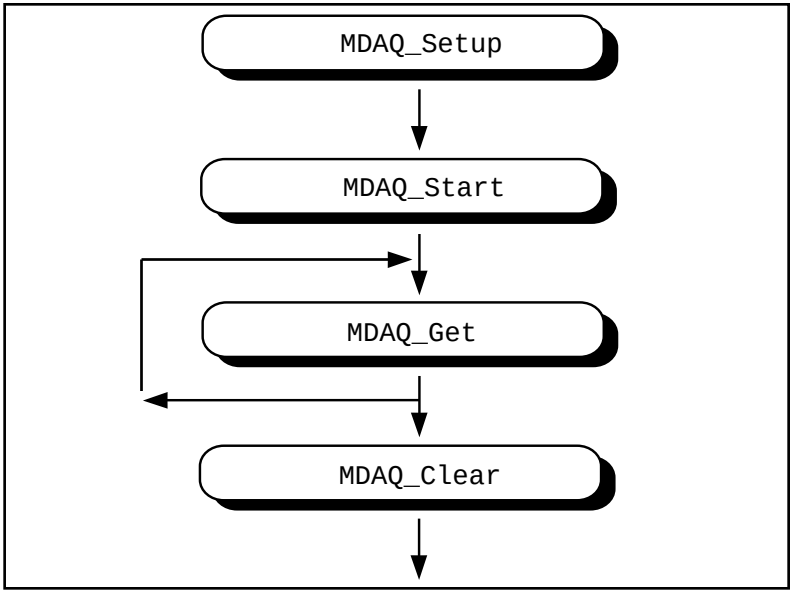


Figure 3-13. Minimum Function Flowchart for an MDAQ Acquisition

`MDAQ_Setup` configures the acquisition buffer. `MDAQ_Start` initiates the data acquisition operation. After starting, a call to `MDAQ_Get` retrieves the frames from the acquisition buffer, and finally `MDAQ_Clear` stops the acquisition and enables successful termination of the application.

Figure 3-14 adds a few functions to the sequence given in Figure 3-13. All functions added to this figure are optional. `MAI_Coupling` can change the coupling default. `MAI_Setup` can select a different combination of channels to scan if not all four channels are needed. `MDAQ_ScanRate` can change the rate of the acquisition.

`MDAQ_Trig_Select` can specify which digital or analog trigger to use. If you specify posttrigger mode in `MDAQ_Setup`, `MDAQ_Trig_Delay` can set up a time delay between the time of the trigger and the beginning of the acquisition.

If you configure the data acquisition to continuously scan or acquire an unlimited number of frames, a call to `MDAQ_Stop` stops the acquisition. After stopping, you can still use `MDAQ_Get` to retrieve any data present on the acquisition buffer, and `MDAQ_Check` returns the progress of the acquisition when `MDAQ_Stop` was called. If you use `MDAQ_Stop`, call it after `MAI_Scale` and before `MDAQ_Clear` in Figure 3-14.

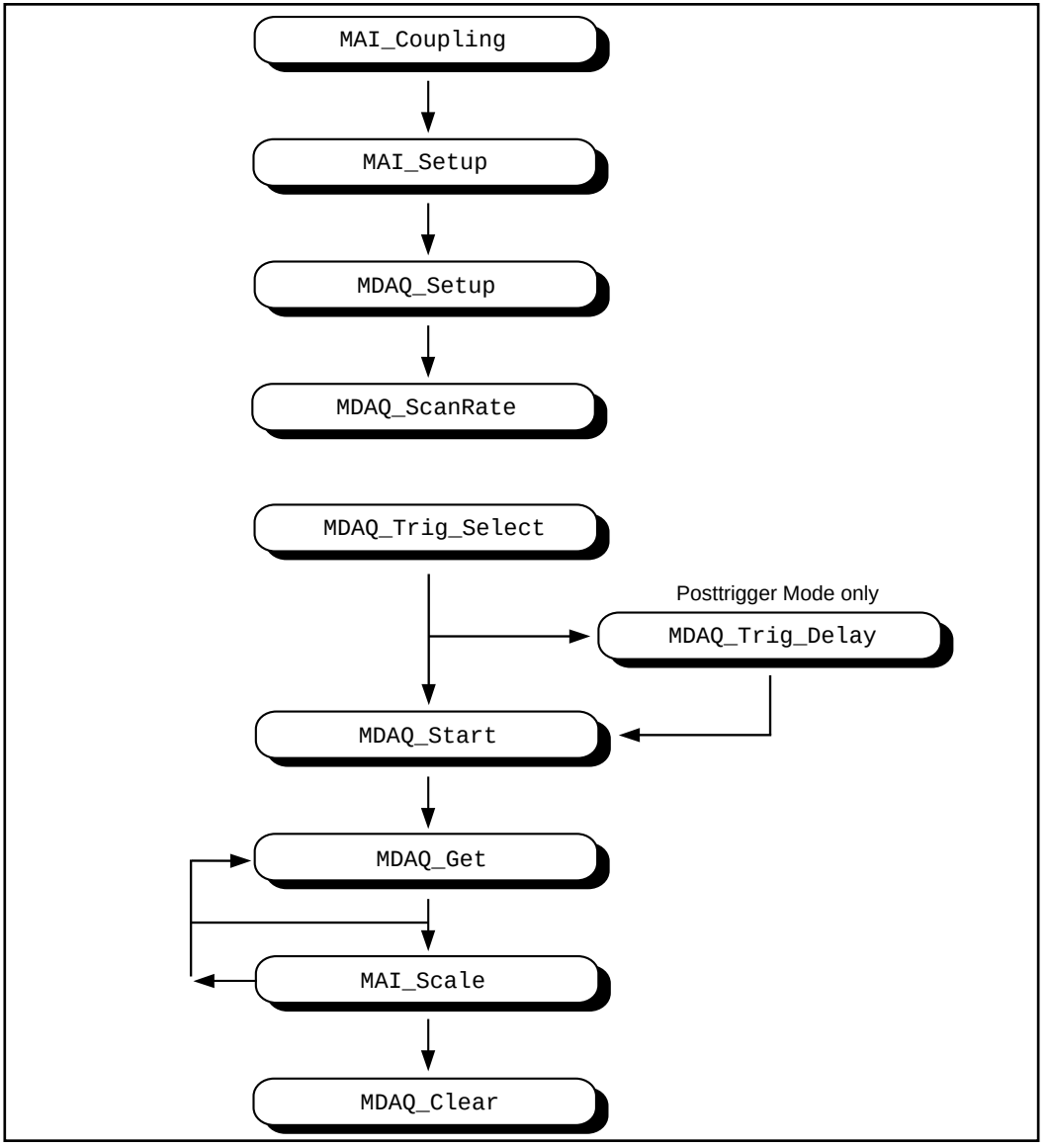


Figure 3-14. MDAQ Acquisition with Optional Coupling and Triggering Configuration

The Analog Output Function Group

The Analog Output function group contains two sets of functions—the Analog Output (AO) functions, which perform single D/A conversions, and the Waveform (WFM) functions, which perform buffered D/A conversions. Both the Analog Output functions and the Waveform functions work with the MIO devices, AT-AO-6/10, Lab-PC+, SCXI-1200, DAQCard-1200, DAQPad-1200 and the AT-DSP2200.

To use the SCXI-1124 analog output module, you must use the SCXI functions.

The Analog Output Functions

Use the Analog Output functions to perform single D/A conversions:

AO_Configure	Records the output range and polarity selected for each analog output channel by the jumper settings on the device and indicates the update mode of the DACs. You must use this function if you have changed the jumper settings affecting analog output range and polarity from their factory settings.
AO_Update	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.
AO_VScale	Scales a voltage to a binary value that, when written to one of the analog output channels, produces the specified voltage.
AO_VWrite	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.

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

Analog Output Application Hints

This section contains a basic explanation of how to construct an application using the analog output functions. The flowcharts are a quick reference for constructing potential applications from the NI-DAQ function calls.

For most purposes, AO_VWrite is the only function required to generate single analog voltages. It converts the floating-point voltage to binary and writes the value to the device. Actually, AO_VWrite is the equivalent of a call to AO_VScale followed by a call to AO_Write. Figure 3-15 illustrates the equivalency.

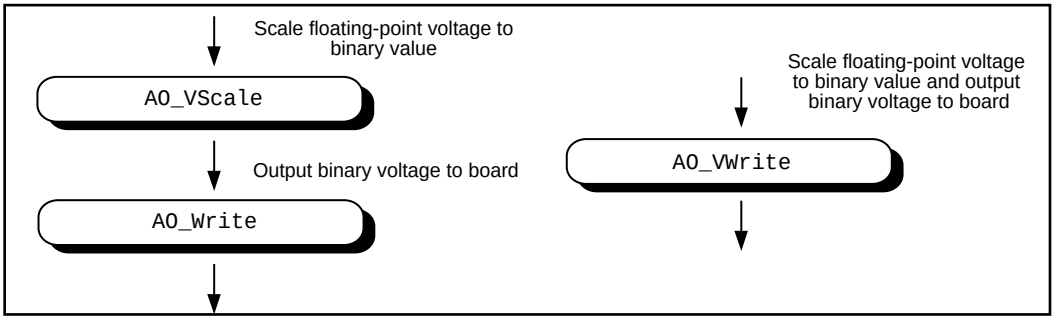


Figure 3-15. Equivalent Analog Output Calls

The following applications are shown using AO_VWrite. However, you could substitute the equivalent AO_VScale and AO_Write calls with no change in results.

Simple Analog Output Application

Figure 3-16 illustrates the basic series of calls for a simple analog output application.

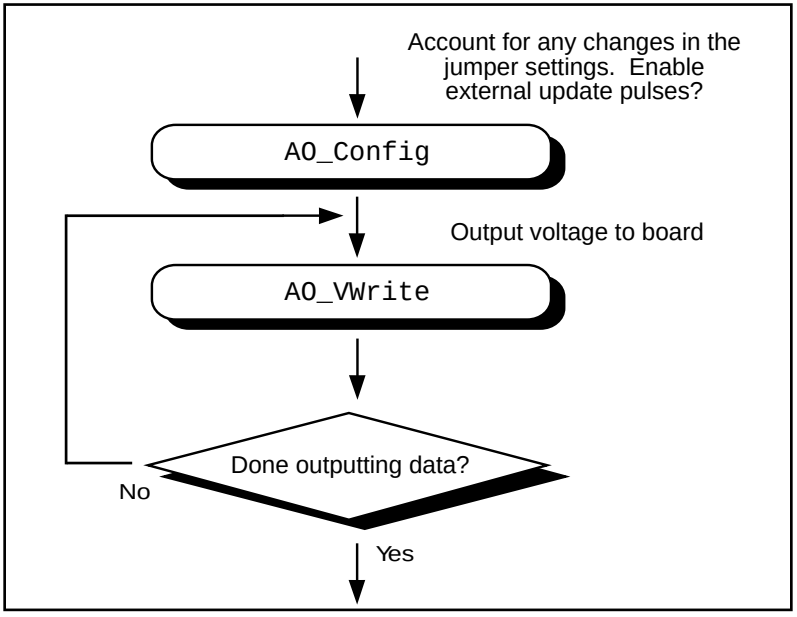


Figure 3-16. Simple Analog Output Application

The call to `AO_Configure` in Figure 3-16 has to be made only if you have changed the jumper settings of an MIO device, AT-AO-6/10, or Lab-PC+. You might also call `AO_Configure` to enable external updating of the voltage. When external update mode is selected, voltages written to the device are not output until you apply a pulse to pin 46 (OUT2) on the I/O connector of the MIO-16 and AT-MIO-16D, to pin 44 (EXTDACUPDATE*) on the AT-MIO-16F-5, AT-MIO-64F-5, and AT-MIO-16X to pin 48 (EXTUPDATE) on the AT-AO-6/10, or to pin 39 (EXTUPDATE) on the Lab-PC+, SCXI-1200, DAQPad-1200, and DAQCard-1200 or to the selected pin on the MIO E Series device. You can simultaneously change the voltages at all the channels.

The final steps in Figure 3-16 form a simple loop. New voltages are output until the end of the data is reached.

Analog Output with Software Update Application

Another application option is to enable later software updates. Like the external update mode, voltages written to the device are not immediately output. Instead, the device does not output the voltages until you call `AO_Update`. In later software update mode, the device changes voltages simultaneously at all the channels. Figure 3-17 illustrates a modified version of the flowchart in Figure 3-16.

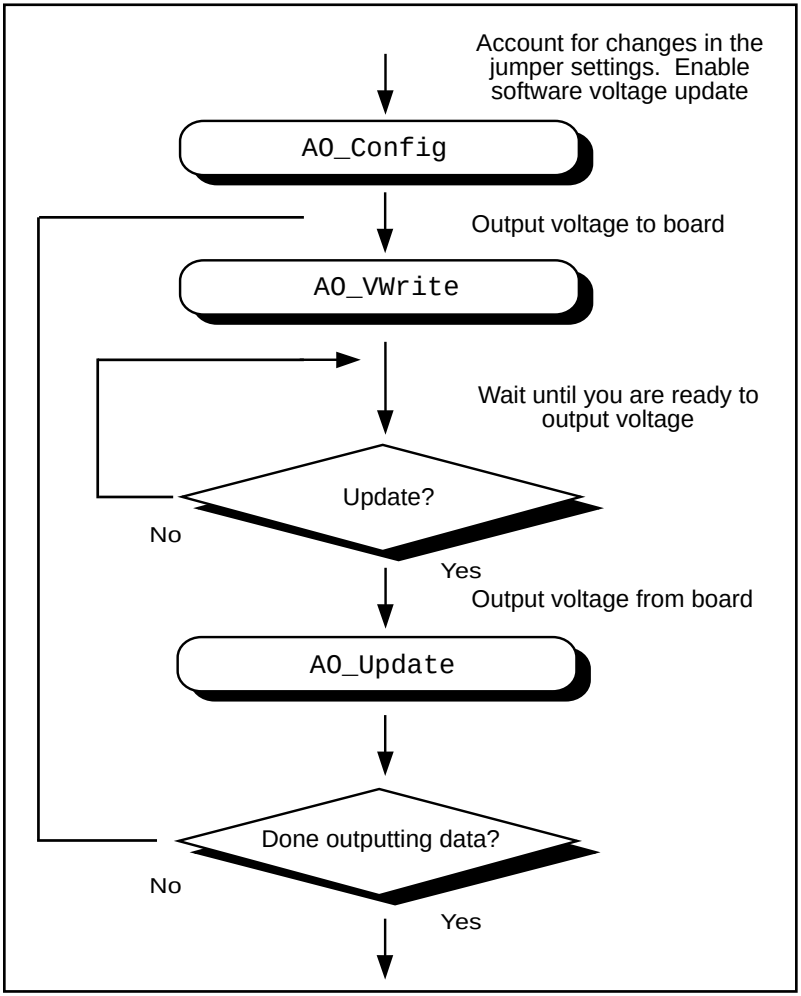


Figure 3-17. Analog Output with Software Updates

The first modification you make is to enable later internal updates when you call AO_Configure. The next change, which follows the AO_VWrite step, is the decision to wait or to output the voltage. If you want the voltage to be output, your application must call AO_Update to write out the voltage. The rest of the flowchart is identical to Figure 3-16.

☞ **Note:** ***Implement buffered analog output via the Waveform Generation functions.***

The Waveform Generation Functions

Use the Waveform Generation functions to perform buffered analog output operations.

The High-Level Waveform Generation Functions

These high-level Waveform Generation functions accomplish with a single call tasks that would require several low-level calls to accomplish:

WFM_from_Disk	Assigns a disk file to one or more analog output channels, selects the rate and number of times the data in the file is to be generated, and starts the generation. WFM_from_Disk waits for completion before returning, unless you call Timeout_Config.
WFM_Op	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.

The Low-Level Waveform Generation Functions

These functions are for setting up, starting, and controlling synchronous Waveform Generation operations:

WFM_Chan_Control	Temporarily halts or restarts waveform generation for a single output channel.

WFM_Check	Returns status information concerning a waveform generation operation.
WFM_ClockRate	Sets an update rate and a delay rate for a group of analog output channels. For the AT-MIO-64F-5 and AT-MIO-16X, this function also sets a delay rate for a group of analog output channels.
WFM_DB_Config	Enables and disables the double-buffered mode of waveform generation.
WFM_DB_HalfReady	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.
WFM_DB_StrTransfer	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.
WFM_DB_Transfer	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).

`WFM_Group_Control`	Controls waveform generation for a group of analog output channels.
`WFM_Group_Setup`	Assigns one or more analog output channels to a waveform generation group. A call to `WFM_Group_Setup` is required only for the AT-AO-6/10. By default, both analog output channels for the Lab-PC+, SCXI-1200, DAQPad-1200, DAQCard-1200, MIO devices, and AT-DSP2200 are in group 1.
`WFM_Load`	Assigns a waveform buffer to one or more analog output channels and indicates the number of waveform cycles to generate. For the AT-MIO-16X, AT-MIO-64F-5, MIO E Series devices, and AT-AO-6/10, this function also enables or disables FIFO mode waveform generation.
`WFM_Rate`	Converts a waveform generation update rate into the timebase and update-interval values needed to produce the desired rate. This function does not support the AT-DSP2200.
`WFM_Scale`	Translates an array of floating-point values that represent voltages into an array of binary values that produce those voltages when the binary array is written to one of the device DACs. The function uses the current analog output configuration settings to perform the conversions.

Waveform Generation Application Hints

This section gives a basic explanation of how to construct an application using the Waveform Generation functions. The flowcharts are a quick reference for constructing potential applications from the NI-DAQ function calls.

Basic Waveform Generation Applications

A basic waveform application outputs a series of voltages to an analog output channel. Figure 3-18 illustrates the ordinary series of calls for a basic waveform application.

The first step of Figure 3-18 calls `WFM_Scale`. The `WFM_Scale` function converts floating-point voltages to integer values which will produce the desired voltages (DAC values).

You have two options available for starting a waveform generation. The first option is to call the high-level function `WFM_Op`. The `WFM_Op` function immediately begins the waveform generation after you call the function. If the number of iterations (repetitions of the buffer) is nonzero, `WFM_Op` does not return until the waveform generation is done and all cleanup work has been completed. Setting the iterations equal to 0 signals NI-DAQ to place the waveform generation in continuous double-buffered mode. In continuous double-buffered mode, waveform generation occurs in the background, and the `WFM_Op` function returns immediately to your application. See the *Double-Buffered Waveform Generation Applications* section later in this chapter for more information.

The second option to start a waveform generation is to call the following sequence of functions:

1. `WFM_Group_Setup` (required only for the AT-AO-6/10) to assign one or more analog output channels to a group
2. `WFM_Load` to assign a waveform buffer to one or more analog output channels
3. `WFM_Rate` to convert a data output rate to a timebase and an update interval that generates the desired rate
4. `WFM_ClockRate` to assign a timebase, update interval, and delay interval to a group of analog output channels. Notice that there are restrictions for using the `WFM_ClockRate` function to specify delay rate. Refer to the `WFM_ClockRate` function description in Chapter 2, *Function Reference*, of the *NI-DAQ Function Reference Manual* for further details.
5. `WFM_Group_Control` (with **operation**=START) to start the waveform generation in the background and return to your application after the waveform generation has begun.

The next step in Figure 3-18 shows the call to `WFM_Check`. `WFM_Check` retrieves the current status of the waveform generation.

Your application uses this information to determine if the generation is complete or should be stopped.

The final step is to call `WFM_Group_Control` (**operation**=CLEAR). The CLEAR operation performs all of the necessary cleanup work after a waveform generation. Additionally, the CLEAR operation halts any ongoing waveform generation.

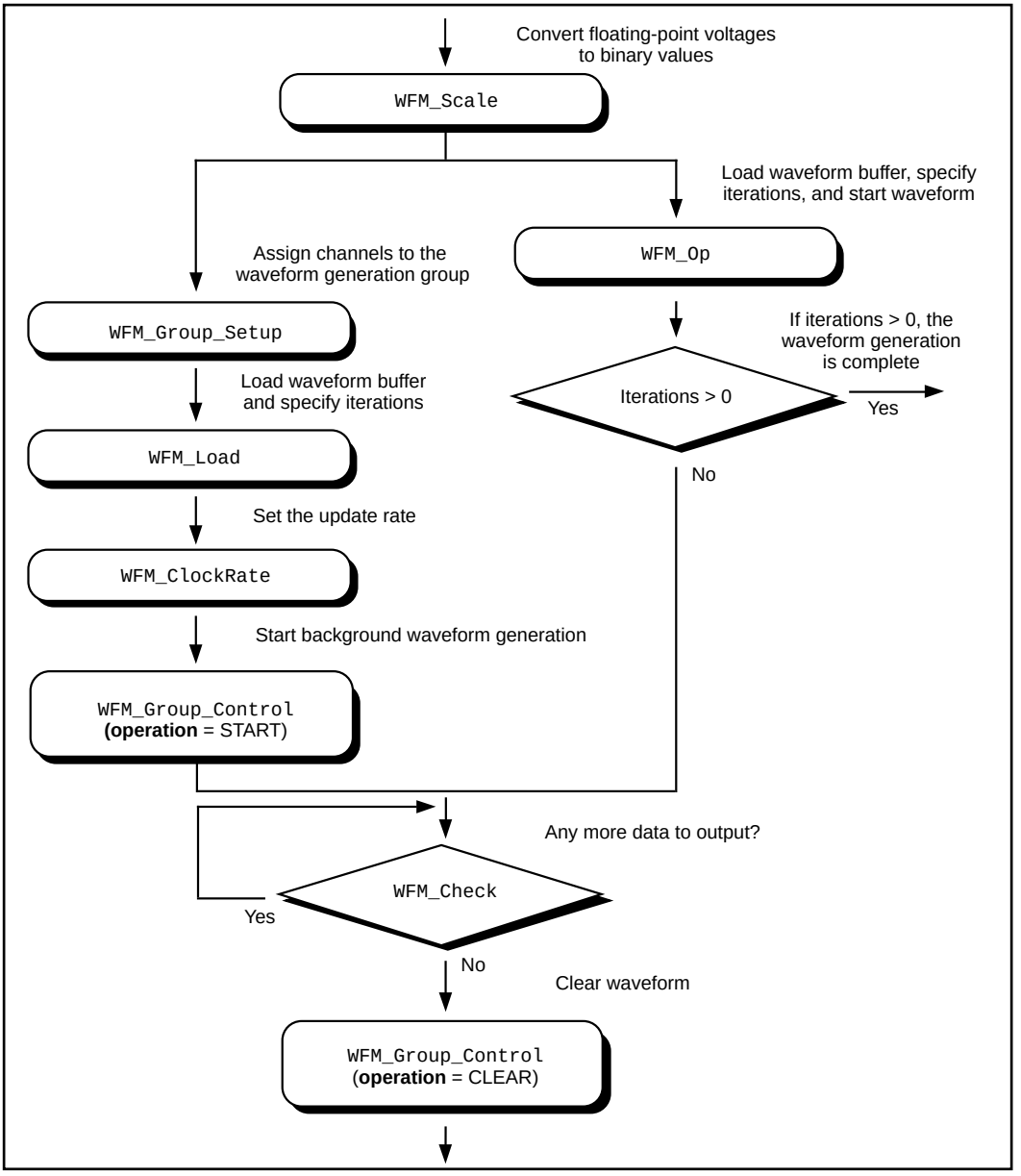


Figure 3-18. Basic Waveform Generation Application

Basic Waveform Generation with Pauses

The application skeleton described in this section is nearly identical to the basic waveform generation application skeleton. The difference is that the description in this section includes the pause and resume operations. Figure 3-19 illustrates the ordinary series of calls for a basic waveform application with pauses.

The first step of Figure 3-19 calls `WFM_Group_Setup`. The `WFM_Group_Setup` function assigns one or more analog output channels to a group.

The second step is to assign a buffer to the analog output channels using the calls `WFM_Scale` and `WFM_Load`. The `WFM_Scale` function converts floating-point voltages to integer values that produce the desired voltages. The `WFM_Load` function assigns a waveform buffer to one or more analog output channels.

The next step is to assign an update rate to the group of channels using the calls `WFM_Rate` and `WFM_ClockRate`. The `WFM_Rate` function converts a data output rate to a timebase and an update interval that generates the desired rate. The `WFM_ClockRate` function assigns a timebase and update interval (and delay interval for the AT-MIO-64F-5 and AT-MIO-16X) to a group of analog output channels.

Notice that there are restrictions for using the `WFM_ClockRate` function to specify delay rate. Refer to the `WFM_ClockRate` function description in Chapter 2, *Function Reference*, of the *NI-DAQ Function Reference Manual*, for further details.

Your application is now ready to start a waveform generation. The `WFM_Group_Control` (**operation**=START) starts the waveform generation in the background. That is, `WFM_Group_Control` returns to your application after the waveform generation has begun.

The next step in Figure 3-19 is an application decision to pause the waveform generation. The application could use a number of conditions for making this decision, including status information returned by `WFM_Check`.

Pause the waveform generation by calling `WFM_Group_Control` (**operation**=PAUSE). The pause operation stops the waveform generation and maintains the current waveform voltage at the channel output.

Resume the waveform generation by calling `WFM_Group_Control` (**operation**=RESUME). The RESUME operation restarts the waveform generation at the data point where it was paused. The output rate and the data buffer are unchanged.

The final step is to call `WFM_Group_Control` (**operation**=CLEAR). The CLEAR operation performs all of the necessary cleanup work after a waveform generation. Additionally, the CLEAR operation halts any ongoing waveform generation.

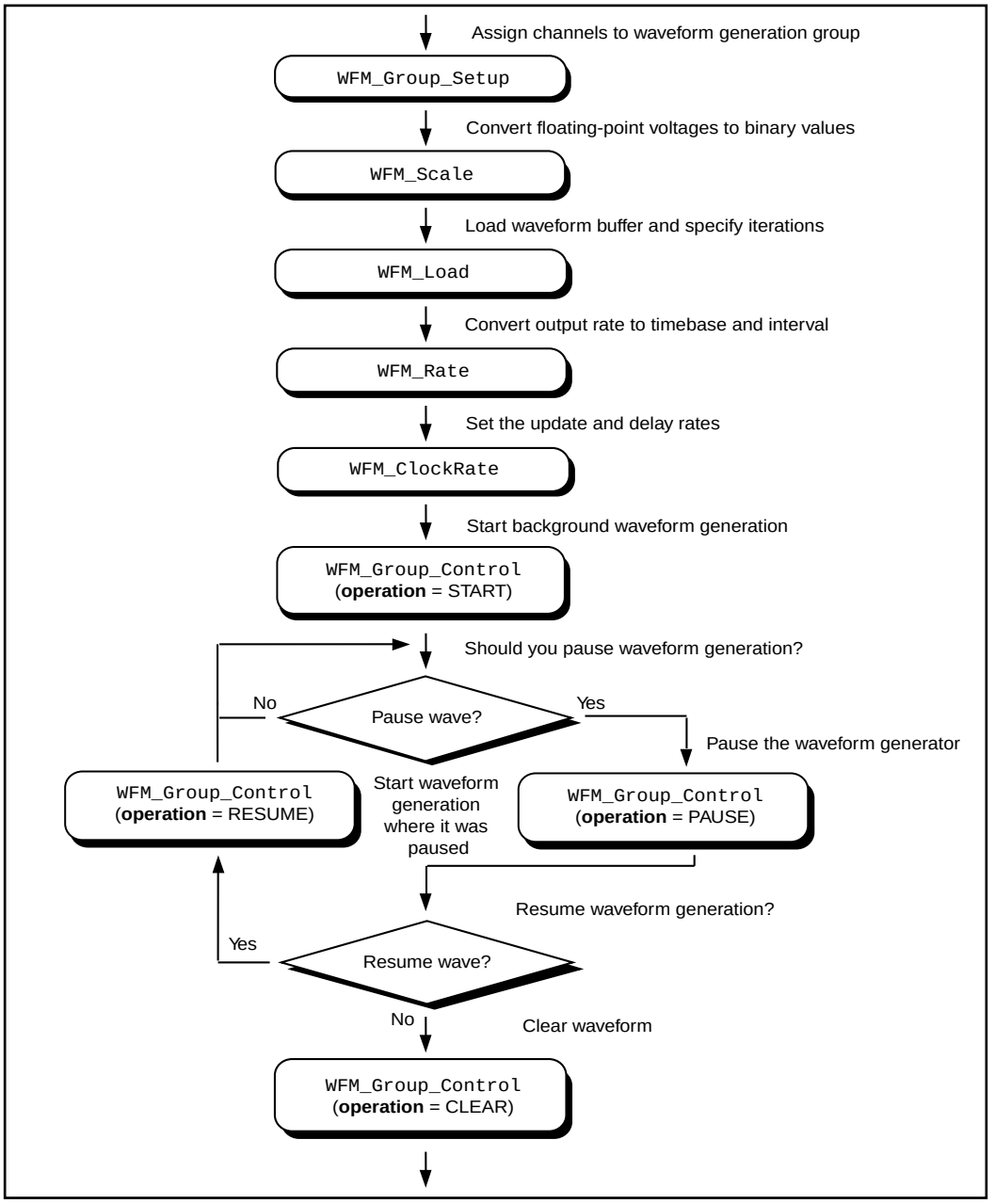


Figure 3-19. Waveform Generation with Pauses

Double-Buffered Waveform Generation Applications

You can also configure waveform generation as a double-buffered operation. Double-buffered operations can perform continuous waveform generation with a limited amount of memory. For an explanation of double buffering, refer to Chapter 5, *NI-DAQ Double Buffering*. Figure 3-20 outlines the basic steps for double-buffered waveform applications.

First, enable double buffering by calling `WFM_DB_Config` as shown in the first step of Figure 3-20.

Although the steps have been left out of the diagram, you might also call `WFM_Rate` and/or `WFM_Scale` as described in the basic waveform application outline.

There are two ways in which your application can start waveform generation. The first way is to call the high-level function `WFM_Op`. The second way to start a waveform generation is to call the following sequence of functions—`WFM_Group_Setup` (only required on the AT-AO-6/10), `WFM_Load`, `WFM_ClockRate`, `WFM_Group_Control` (**operation**=START). The `WFM_Group_Setup` function assigns one or more analog output channels to a group. The `WFM_Load` function assigns a waveform buffer to one or more analog output channels. This buffer is called a circular buffer. The `WFM_ClockRate` function assigns a timebase and update interval to a group of analog output channels. The `WFM_Group_Control` (**operation**=START) starts the waveform generation in the background. That is, `WFM_Group_Control` returns to your application after the waveform generation has begun.

After the operation has started, you can perform any number of transfers to the circular waveform buffer. To transfer data to the circular buffer, call the `WFM_DB_Transfer` function. After the function is called, NI-DAQ will wait until it is able to transfer the data before returning to the application. To avoid the waiting period, you can call `WFM_DB_HalfReady` to determine if the transfer can be made immediately. If `WFM_DB_HalfReady` indicates NI-DAQ is not ready for a transfer, your application is free to do other processing and check the status later.

After the final transfer, you may want to call `WFM_Check` to get the current progress of the transfer. Remember, NI-DAQ requires some time after the final transfer to actually output the data.

The final step is to call `WFM_Group_Control` (**operation**=CLEAR). The CLEAR operation performs all of the necessary cleanup work after a waveform generation. Additionally, the CLEAR operation will halt any ongoing waveform generation.

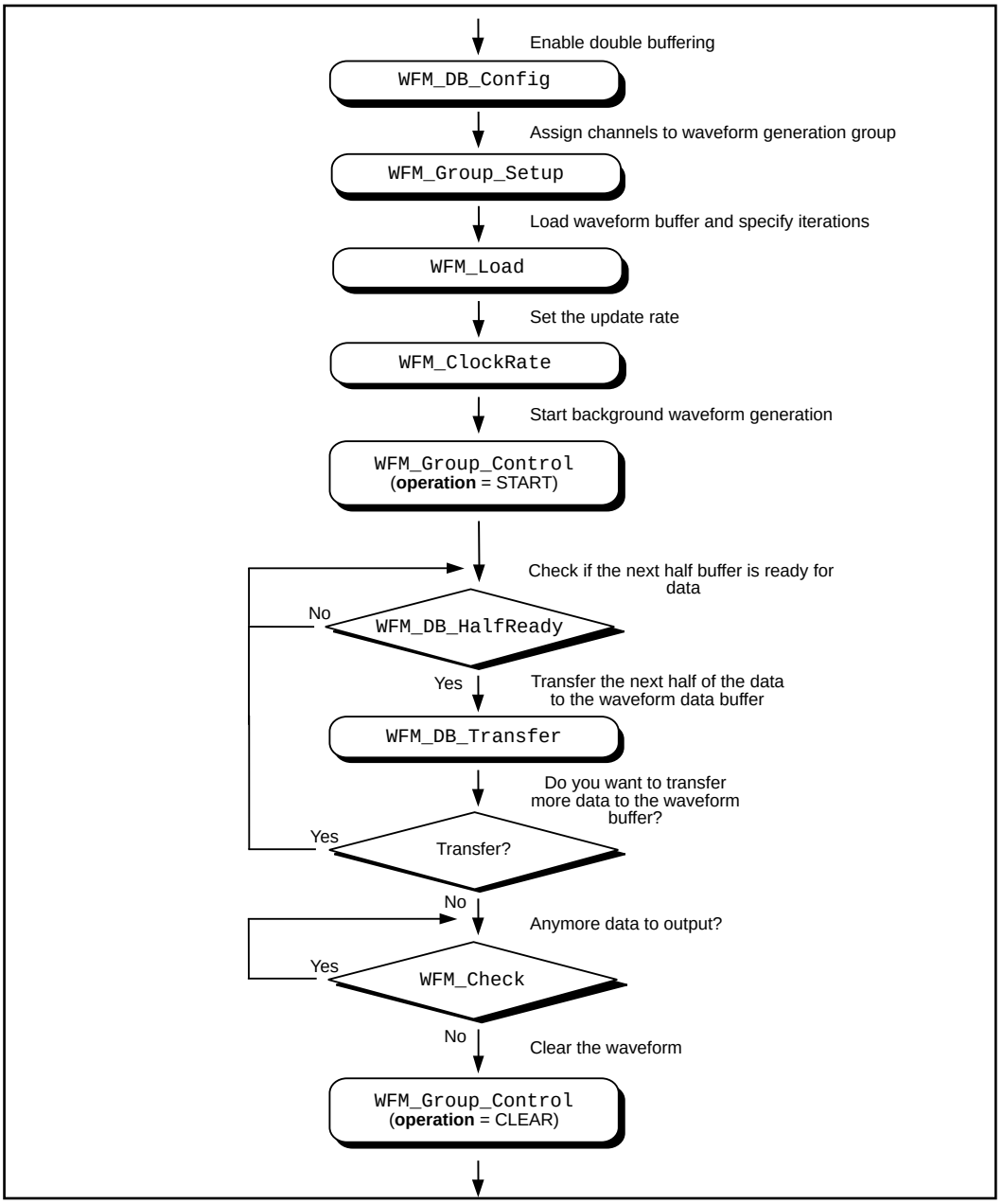


Figure 3-20. Double-Buffered Waveform Generation

Reference Voltages for Analog Output Devices

The following table shows the output voltages produced when you select unipolar output polarity.

Table 3-4. Output Voltages with Unipolar Output Polarity

Device	Value in Waveform Buffer	
	0	**4,095**
AT-MIO-16X	0 V	65,535
All Other MIO Devices	0 V	reference voltage
AT-AO-6/10	0 V	reference voltage (+10 V in default case)
Lab-PC+, SCXI-1200, DAQPad-1200, DAQCard-1200	0 V	+5 V

The following table shows the output voltages produced when you select bipolar output polarity.

Table 3-5. Output Voltages with Bipolar Output Polarity

Device	Value in Waveform Buffer			
	-2,048	**2,047**	**-32,768**	**32,767**
AT-MIO-16X	—	—	negative of the reference voltage	reference voltage
All Other MIO Devices	negative of the reference voltage	reference voltage	—	—
AT-AO-6/10	negative of the reference voltage (-10 V in default case)	reference voltage (+10 V in default case)	—	—

Table 3-5. Output Voltages with Bipolar Output Polarity (Continued)

Device	Value in Waveform Buffer			
	-2,048	**2,047**	**-32,768**	**32,767**
Lab-PC+, SCXI-1200, DAQPad-1200, DAQCard-1200	-5 V	+5 V	—	—
AT-DSP2200	—	—	-2.828 V	+2.828 V

Minimum Update Intervals

The rate at which a device can output analog data is limited by the performance of the host computer. For waveform generation, the limitation is in terms of minimum update intervals. The update interval is the period of time between outputting new voltages. Therefore, the minimum update interval specifies the smallest possible time delay between outputting new data points. In other words, the minimum update interval specifies the fastest rate at which a device can output data. Refer to Chapter 4, *DMA and Programmed I/O Performance Limitations*, for more information on the minimum update intervals.

Notes on DMA Waveform Generation with the AT-MIO-16F-5

Page breaks in the buffer can adversely affect DMA waveform generation of the AT-MIO-16F-5. Page breaks are described in Chapter 4, *DMA and Programmed I/O Performance Limitations*.

Use the utility function `Align_DMA_Buffer` to avoid the negative effects of page boundaries on PC AT and compatible computers in the following cases:

- When using DMA waveform generation at update intervals faster than about 50 µs (this number will depend on your PC).
- When using interleaved DMA waveform generation

To use `Align_DMA_Buffer`, you must allocate a buffer that is bigger than the sample count to give `Align_DMA_Buffer` room to move the data around. If you are using interleaved DMA waveform generation but at rates that can tolerate a page break, allocating an extra two bytes is sufficient to position the data so that the page break does not cause unpredictable results. After the buffer is aligned, you can

make the normal calls to all of the Waveform Generation functions. If you need to access the data in an aligned buffer while the waveform is in progress, use the index that `Align_DMA_Buffer` returned. When you make a call to `WFM_Group Control` (**operation**=CLEAR), or make a call to `WFM_Load` with a new buffer while the waveform is in progress, NI-DAQ *unaligns* the previous waveform buffer. If you want to use the same buffer again for waveform generation after it has been unaligned, you must call `Align_DMA_Buffer` again. See the function description for `Align_DMA_Buffer` in Chapter 2, *Function Reference*, of the *NI-DAQ Function Reference Manual* for more information.

The first point of the waveform buffer is not generated until the second update pulse occurs. That is, two update intervals will pass before the waveform begins to appear at the I/O connector after a `WFM_Op`, `WFM_from_Disk`, or `WFM_Group_Control` (**operation**=START) call starts the waveform.

When you use double DMA waveform generation on the AT-MIO-16F-5—that is, when waveforms are generated at both DACs and a separate DMA channel services each DAC—the two waveforms must terminate simultaneously. You should either set up both channels to terminate after generating the same number of points, or set up both channels to generate indefinite waveforms. If, for example, you set up channel 0 to terminate after 10 iterations through its buffer and set up channel 1 for indefinite waveform generation, when channel 0 has finished its iterations, it forces channel 1 to stop generating points also. This restriction does not apply to double waveform generation that is interrupt driven.

Counter Usage

For the MIO E Series devices dedicated counters from the DAQ-STC chip are used for control and timing of waveform generation.

For the MIO-16 and AT-MIO-16D, counter 2 generates interrupt requests that result in the voltage updates at the analog output channels. If counter 2 is otherwise in use (such as for interval scanning), waveform generation is not possible until counter 2 is freed.

For the AT-MIO-16F-5, AT-MIO-64F-5, and AT-MIO-16X, counters 1, 2, and 5, as well as the dedicated external update signal, can generate either interrupt or DMA requests. If it is available, NI-DAQ uses

counter 5 for waveform generation. Otherwise, NI-DAQ uses counter 2. If counter 2 is also unavailable, NI-DAQ selects counter 1.

On the Lab-PC+, SCXI-1200, DAQPad-1200, and DAQCard-1200, counter A2 produces the total update interval for waveform generation. However, if the total update interval is greater than 65,535 µs, counter B0 generates the clock for a slower timebase, which counter A2 uses for the total update interval. The `ICTR_Setup` and `ICTR_Reset` functions cannot then use counter B0 for the duration of the waveform generation operation. In addition, the Data Acquisition functions `DAQ_Start` and `Lab_ISCAN_Start` cannot use counter B0 if the total sample interval for data acquisition is also greater than 65,535 µs, unless the timebase required for data acquisition is the same as the timebase counter B0 produces for waveform generation. If data acquisition is not in progress, counter B0 is available for waveform generation if `ICTR_Setup` has not been called on counter B0 since startup, or an `CTR_Reset` call has been made on counter B0. If data acquisition is in progress and is using counter B0 to produce the sample timebase, counter B0 is available for waveform generation only if this timebase is the same as required by the Waveform Generation functions to produce the total update interval. In this case, counter B0 provides the same timebase for data acquisition and waveform generation.

On the AT-AO-6/10, counter 0 produces the total update interval for group 1 waveform generation and counter 1 produces the total update interval for group 2 waveform generation. However, if the total update interval is greater than 65,535 µs for either group 1 or 2, counter 2 is used by counter 0 (group 1) or counter 1 (group 2) to provide the total update interval. If either group is using counter 2 to produce the sample timebase, counter 2 is available to the other group only if the timebase is the same as the timebase required by the Waveform Generation functions to produce the total update interval. In this case, counter 2 provides the same timebase for both waveform generation groups.

Restrictions on the Use of a Delay Rate on the AT-MIO-16X and AT-MIO-64F-5

To set a delay rate using `WFM_ClockRate`, the entire waveform buffer must fit within the analog output FIFOs of these devices. The FIFOs on both devices can hold 2,048 samples. The number of iterations of the buffer must be greater than zero and less than or equal to 65,535. Also, double-buffered waveform generation is incompatible with the use of a delay rate.

FIFO Lag Effect on the MIO E Series, AT-AO-6/10, AT-MIO-16X, and AT-MIO-64F-5

Group 1 analog output channels use an onboard FIFO to output data values to the DACs. NI-DAQ continuously writes values to the FIFO as long as the FIFO is not full. NI-DAQ transfers data values from the FIFO to the DACs at regular intervals using an onboard or external clock. A lag effect is seen for group 1 channels due to the buffering of the FIFO. That is, a value written to the FIFO is not output to the DAC until all of the data values currently in the FIFO have been output to the DACs. This time lag is dependent upon the update rate (specified in `WFM_ClockRate`). Refer to your device user manual for a more detailed discussion of the onboard FIFO.

Three functions are affected by the FIFO lag effect—`WFM_Chan_Control`, `WFM_Check`, and double-buffered waveform generation.

`WFM_Chan_Control`—When you execute **operation**=PAUSE for a group 1 channel, the effective pause does 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 been emptied.

`WFM_Check`—The values returned in **pointsDone** and **itersDone** indicate the number of points that NI-DAQ has written to the FIFO for the specified channel. A time lag occurs from the point when NI-DAQ writes the data to the FIFO when NI-DAQ outputs the data to the DAC. The **status** parameter is also affected by the FIFO lag because **status** indicates when NI-DAQ writes the last point to the FIFO.

When you use double-buffered waveform generation with group 1, make sure the total number of points for all of the group 1 channels (specified in the **count** parameter in `WFM_Load`) is at least twice the size of the FIFO. Refer to your device user manual for information on the AT-AO-6/10 FIFO size.

Using Onboard AT-DSP2200 Memory

The AT-DSP2200 can generate waveforms from buffers that reside on its own onboard memory. Use the NI-DAQ Memory Management function, `NI_DAQ_Mem_Alloc`, as well as memory allocation calls in the NI-DSP library, to allocate onboard AT-DSP2200 buffers. You

can use the memory handles returned by these calls in place of the buffer parameters in the WFM_Op and WFM_Load calls.

Externally Triggering Your Waveform Generation Operation

It is possible to initiate a waveform generation operation from an external trigger signal in much the same manner as for analog input. For MIO E Series devices, see the Select_Signal function description in the *NI-DAQ Function Reference Manual for PC Compatibles*.

On Am9513-based devices, you need to call CTR_Config and change the gating mode before you call the WFM functions. Refer to the *Using This Function* section of the CTR_Config function description in the *NI-DAQ Function Reference Manual for PC Compatibles*.

The Digital I/O Function Group

The Digital I/O function group contains three sets of functions—the Digital I/O (DIG) functions, the Group Digital I/O (DIG_Block, DIG_Grp, and DIG_SCAN) functions, and the double-buffered Digital I/O (DIG_DB) functions.

The following devices contain digital I/O hardware:

- All DIO devices
- All MIO and AI devices
- AT-AO-6/10
- DAQCard-500, DAQCard-700, PC-LPM-16
- Lab-PC+, SCXI-1200, DAQPad-1200, and DAQCard-1200
- PC-TIO-10
- AO-2DC devices

To use the SCXI digital and relay modules, you can use the SCXI functions.

These devices contain a number of digital I/O ports of up to eight digital lines in width. The name *port*, in fact, refers to a set of digital lines (digital lines are also referred to as bits in this text). In many instances, you control the set of digital lines as a group for both reading and writing purposes and for configuration purposes. For example, you can configure the port as an input port or as an output port, which

means that the set of digital lines making up the port consist of either all input lines or all output lines.

The digital ports are usually assigned letters, and the digital lines making up the port are assigned numbers beginning with 0. For example, the DIO-24 contains three ports of eight digital lines each. These ports are labeled PA, PB, and PC on the DIO-24 I/O connector drawing, as shown in the *PC-DIO-24 User Manual*. The eight digital lines making up port PA are labeled PA7 through PA0.

In some cases, you can further combine digital I/O ports into a larger entity called a *group*. On the DIO-32F, for example, you can assign any of its ports DIOA through DIOD to one of two groups. A group of ports are handshaked or clocked as a unit.

The Digital I/O functions can write to and read from both an entire port and single digital lines within the port. To write to an entire port, NI-DAQ writes a byte of data to the port in a specified digital output pattern. To read from a port, NI-DAQ returns a byte of data in a specified digital output pattern. The mapping of the byte to the digital I/O lines is as follows:

Bit Number	Digital I/O Line Number
7	7 Most significant bit
6	6
5	5
4	4
3	3
2	2
1	1
0	0 Least significant bit

In the cases where a digital I/O port has fewer than eight lines, the most significant bits in the byte format are ignored.

You can configure most of the digital I/O ports as either input ports or output ports. On the PC-TIO-10, you can independently program lines on the same port as input or output lines. Some digital I/O ports are permanently fixed as either input ports or output ports. If you configure a port as an input port, reading that port returns the value of the digital lines. In this case, external devices connected to and driving those lines determine the state of the digital lines. If no external device is driving the lines, the lines float to some indeterminate state, and you can read them in either state 0 (digital logic low) or state 1 (digital logic high). If you configure a port as an output port, writing to the port sets each digital line in the port to a digital logic high or low, depending on the data written. In this case, these digital lines can drive an external device. Many of the digital I/O ports have read-back capability; that is, if you configure the port as an output port, reading the port returns the output state of that port.

You can use digital I/O ports on the DIO-24, AT-MIO-16D, AT-MIO-16DE-10, PC-DIO-96, Lab-PC+, SCXI-1200, DAQPad-1200, and DAQCard-1200 and the groups on the DIO-32F for handshaking modes. For the remainder of this chapter, *no-handshaking mode* is synonymous with nonlatched mode, and *handshaking mode* is synonymous with latched mode. These two modes have the following characteristics:

- No-handshaking (nonlatched) mode—This mode simply changes the digital value at an output port when written to and returns a digital value from a digital input port when read from. No handshaking signals are generated.
- Handshaking (latched) mode—You can use this mode for digital I/O handshaking; that is, a digital input port latches the data present at the input when the port receives a handshake signal and generates a handshake pulse when the computer writes to a digital output port. In this mode, you can read the status of a port or a group of ports to determine whether an external device has accepted data written to an output port or has latched data into an input port.

Process control applications, such as controlling or monitoring relays, often use the no-handshaking mode. Communications applications, such as transferring data between two computers, often use the handshaking mode.

DIO-24, AT-MIO-16D, AT-MIO-16DE-10, PC-DIO-96, Lab-PC+, SCXI-1200, DAQPad-1200, and DAQCard-1200 Groups

You can group together any combination of ports 0, 1, 3, 4, 6, 7, 9, and 10 on the PC-DIO-96, of ports 0 and 1 on the DIO-24, Lab-PC+, SCXI-1200, DAQPad-1200, and DAQCard-1200, and of ports 2 and 3 on the AT-MIO-16D and AT-MIO-16DE-10 to make up larger ports. For example, with the DIO-96 you can program ports 0, 3, 9, and 10 to make up a 32-bit handshaking port, or program all eight ports to make up a 64-bit handshaking port. See *Digital I/O Application Hints* later in this chapter and the `DIG_SCAN_Setup` function in Chapter 2, *Function Reference*, of the *NI-DAQ Function Reference Manual* for more details.

DIO-32F Groups

On the DIO-32F, you can assign ports 0 to 3 to one of two groups for handshaking. These groups are referred to as group 1 and group 2. Group 1 uses handshake lines REQ1 and ACK1. Group 2 uses handshake lines REQ2 and ACK2. The group senses the REQ line. An active REQ signal is an indication that the group must perform a read or write. The group drives the ACK line. After the group has performed a read or write, it drives the ACK line to its active state. Refer to the *AT-DIO-32F User Manual* for more information on the handshaking signals.

Group 1 can be 8, 16, or 32 bits wide. If you set group 1 to eight bits, you have assigned either port 0 or port 1 to the group. If you set group 1 to 16 bits, you have assigned both port 0 and port 1 to the group, and read and write operations are addressed to port 0. If you set group 1 to 32 bits, then you have assigned all four ports to the group. You can only use a 32-bit group for buffered digital I/O.

Group 2 can be 8 or 16 bits wide. If you set group 2 to eight bits, you have assigned either port 2 or port 3 to the group. If you set group 2 to 16 bits, you have assigned both port 2 and port 3 to the group, and read and write operations are addressed to port 2.

After you have assigned ports to a group, the group acts as a single entity controlling 8, 16, or 32 digital lines simultaneously. The following assignments are the legal group assignments.

Assigned Ports	Group Name	Group Size (in Bits and Ports)	
port 0	1	8-bit group	one port
port 1	1	8-bit group	one port
port 2	2	8-bit group	one port
port 3	2	8-bit group	one port
ports 0 and 1	1	16-bit group	two ports
ports 2 and 3	2	16-bit group	two ports
ports 0, 1, 2, and 3	1	32-bit group	four ports

After ports are assigned to a group, the group controls handshaking of that port. These ports are then read from or written to simultaneously by writing or reading 8 or 16 bits at one time from the group.

You can configure the groups for various handshake configurations. The configuration choices include level or edge-triggered handshaking, inverted or non-inverted ACK and REQ lines, and a programmed transfer settling time.

☞ **Note:** ***Implement buffered digital I/O via the*** `DIG_Block` ***functions described in detail in Chapter 2,*** *Function Reference****, of the*** *NI-DAQ Function Reference Manual****.***

The Digital I/O Functions

The Digital I/O (`DIG`) functions perform nonhandshaked digital line and port I/O for all DIO, MIO, and AI devices, the AT-AO-6/10, Lab-PC+, PC-LPM-16, DAQCard-500, DAQCard-700, PC-TIO-10, SCXI-1200, DAQPad-1200, DAQCard-1200, and AO-2DC devices. The Digital I/O functions also perform handshaked port I/O for the

DIO-24, AT-MIO-16D, AT-MIO-16DE-10, Lab-PC+, PC-DIO-96, SCXI-1200, DAQPad-1200, and DAQCard-1200:

`DIG_In_Line`	Returns the digital logic state of the specified digital input line in the specified port.
`DIG_In_Port`	Returns digital input data from the specified digital I/O port.
`DIG_Line_Config`	Configures the specified line on a specified port for direction (input or output) for the PC-TIO-10 and E Series devices only.
`DIG_Out_Line`	Sets or clears the specified digital output line in the specified digital port.
`DIG_Out_Port`	Writes digital output data to the specified digital port.
`DIG_Prt_Config`	Configures the specified port for direction (input or output).
`DIG_Prt_Status`	Returns a status word indicating the handshake status of the specified port (DIO-24, AT-MIO-16D, AT-MIO-16DE-10, PC-DIO-96, Lab-PC+, SCXI-1200, DAQPad-1200, and DAQCard-1200 only).

The Group Digital I/O Functions

The Group Digital I/O (`DIG_Block`, `DIG_Grp`, and `DIG_SCAN`) functions perform handshaked I/O on groups of ports for the DIO-24, PC-DIO-96, Lab-PC+, AT-MIO-16D, AT-MIO-16DE-10, AT-DIO-32F, SCXI-1200, DAQPad-1200, and DAQCard-1200:

`DIG_Block_Check`	Returns the number of items remaining to be transferred after a `DIG_Block_In` or `DIG_Block_Out` call.

`DIG_Block_Clear`	Halts any ongoing asynchronous transfer, allowing another transfer to be initiated.
`DIG_Block_In`	Initiates an asynchronous transfer of data from the specified group to memory.
`DIG_Block_Out`	Initiates an asynchronous transfer of data from memory to the specified group.
`DIG_Block_PG_Config`	Enables or disables the pattern generation mode of buffered digital I/O (DIO-32F only).
`DIG_Grp_Config`	Configures the specified group for port assignment, direction (input or output), and size (DIO-32F only).
`DIG_Grp_Mode`	Configures the specified group for handshake signal modes (DIO-32F only).
`DIG_Grp_Status`	Returns a status word indicating the handshake status of the specified group (DIO-32F only).
`DIG_In_Grp`	Reads digital input data from the specified digital group (DIO-32F only).
`DIG_Out_Grp`	Writes digital output data to the specified digital group (DIO-32F only).
`DIG_SCAN_Setup`	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, Lab-PC+, SCXI-1200, DAQPad-1200, and DAQCard-1200 only).

The Double-Buffered Digital I/O Functions

The double-buffered Digital I/O (DIG) functions perform double-buffered operations during Group Digital I/O operations:

DIG_DB_Config	Enables or disables double-buffered digital transfer operations and sets the double-buffered options.
DIG_DB_HalfReady	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 the data can be transferred before returning.
DIG_DB_StrTransfer	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 is passed 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 data is saved on disk as it is acquired. You can then write the string to a disk file using the BASIC PUT statement.
DIG_DB_Transfer	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 is passed 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.

Digital I/O Application Hints

This section gives a basic explanation of how to construct an application using the digital input and output functions. The flowcharts are a quick reference for constructing potential applications from the NI-DAQ function calls.

Latched Versus Nonlatched Digital I/O

Digital ports can output or input digital data in two ways. The first is to immediately read or write data to or from the port. This type of digital I/O is called nonlatched. The second method is to coordinate digital data transfers with another digital port. The second method is called latched I/O or digital I/O with handshaking. With handshaking, you use dedicated transmission lines to ensure that data on the receiving end is not overwritten with new data before it can be read from the input port.

NI-DAQ supports both latched and nonlatched modes. The application outlines within this section explain the use of both modes where they apply.

Digital Port I/O Applications

Digital port I/O applications use individual digital ports to input or output digital data. In addition, the applications input or output data points on an individual basis.

Individual port transfers can be latched or nonlatched. All AT Series and PC Series devices can use nonlatched digital port I/O. DIO-24, AT-MIO-16D, AT-MIO-16DE-10, PC-DIO-96, Lab-PC+, SCXI-1200, DAQPad-1200, and DAQCard-1200 can also use latched digital I/O for these purposes.

Figure 3-21 illustrates the ordinary series of calls for digital port I/O applications.

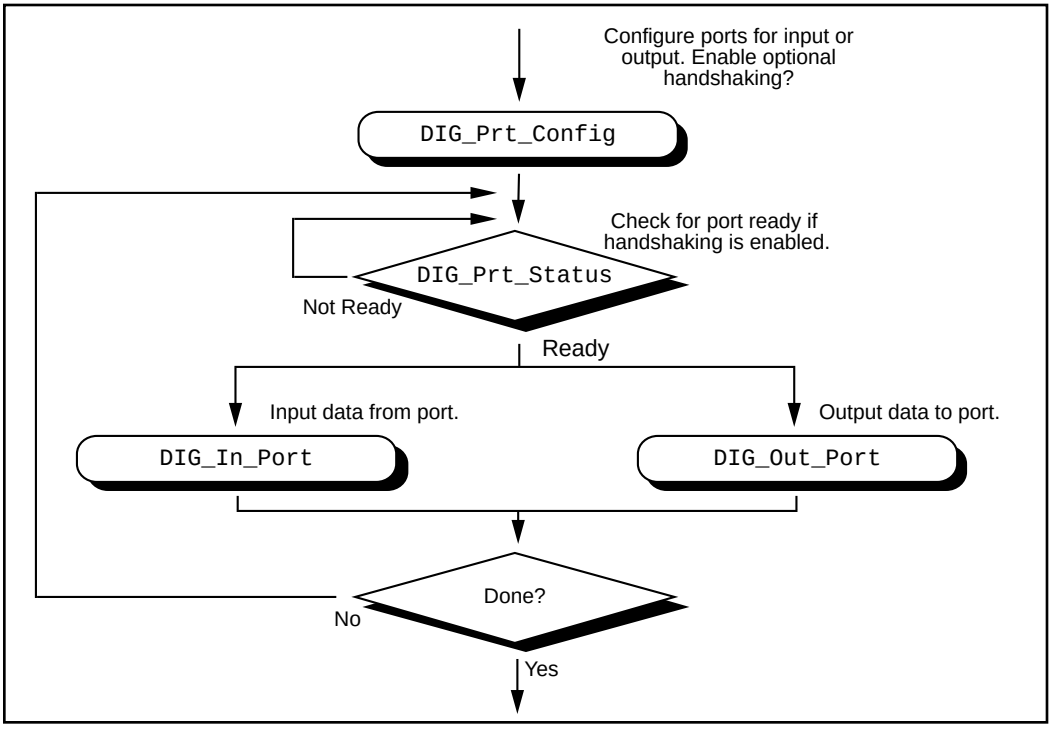


Figure 3-21. Basic Port Input or Output Application

The first step is to call `DIG_Prt_Config`, with which you configure individual digital ports for input or output. Also, you use `DIG_Prt_Config` to enable handshaking.

If handshaking is disabled, do not check the port status (step 2 of Figure 3-21). If handshaking is enabled, you should call `DIG_PRT_Status` to determine if an output port is ready to output a new data point, or if an input port has latched new data.

The third step is to actually input or output the data point. Call `DIG_In_Port` to read data from an input port. Call `DIG_Out_Port` to write data to port.

The final step is to loop back if more data is to be input or output. These four steps form the basis of a simple digital port I/O application.

Digital Line I/O Applications

Digital line I/O applications are similar to digital port I/O applications, except that digital line I/O applications input or output data on a bit-by-bit basis rather than by port. The digital line I/O can only transfer data in no-handshaking mode.

Figure 3-22 is a flowchart outlining the basic line I/O application.

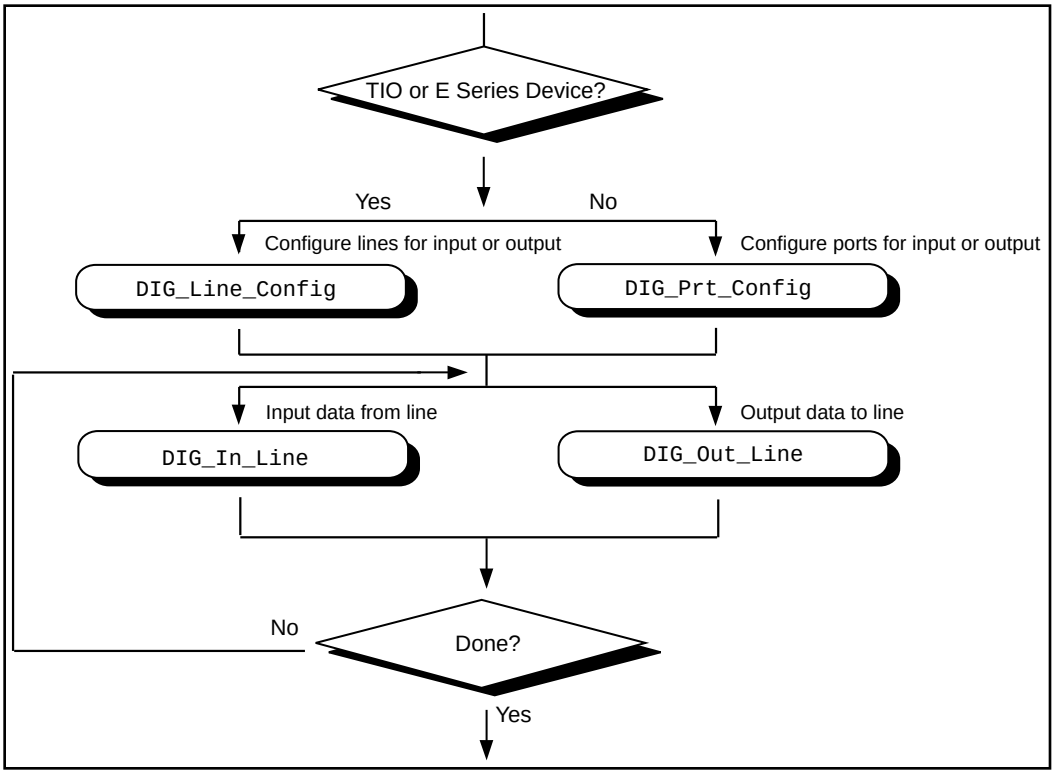


Figure 3-22. Basic Line Input or Output Application

The first step is to configure the digital lines for input or output. You can program PC-TIO-10 and E Series devices on an individual line basis. To do this, call `DIG_Line_Config`. You must configure all other devices on a port-by-port basis, however. As a result, you must configure all lines within a port for the same direction. Call `DIG_Prt_Config` to configure a port for input or output.

The next step is to call `DIG_In_Line` or `DIG_Out_Line` to output or input a bit from or to the line. The final step is to loop back until NI-DAQ has transferred all of the data.

Digital Group I/O Applications

Digital group I/O applications use one or more digital ports as a single group to input or output digital information.

Figure 3-23 is a flowchart for group digital applications that input and output data on a point-by-point basis. Only the DIO-32 boards can group input or output on a point-by-point basis.

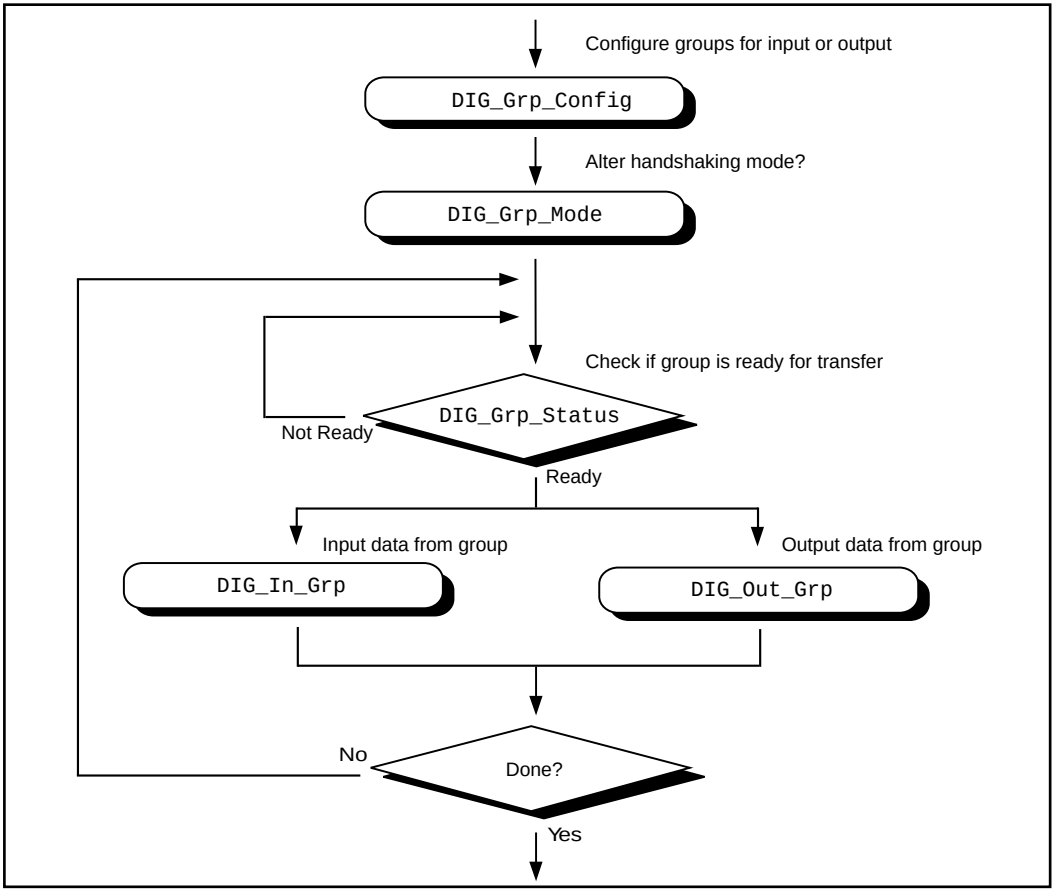


Figure 3-23. Simple Digital Group Input or Output Application

At the start of your application, you must call `DIG_Grp_Config` to configure the individual digital ports as a group. After the ports are grouped, you can alter the handshaking mode of the DIO-32 by calling `DIG_Grp_Mode` (step 2 of Figure 3-23). The various handshaking modes and the default settings are explained in the `DIG_Grp_Mode` function description.

The next step in your application is to check if the port is ready for a transfer (step 3 of Figure 3-23). To do this, call `DIG_Grp_Status`. If the group status indicates it is ready, call `DIG_Out_Grp` or `DIG_In_Grp` to transfer the data to or from the group.

The final step of the flowchart is to loop back until all of the data has been input or output.

Digital Group Block I/O Applications

NI-DAQ also contains group digital I/O functions, which operate on blocks of data. Figure 3-24 outlines the basic steps for applications that use block I/O.

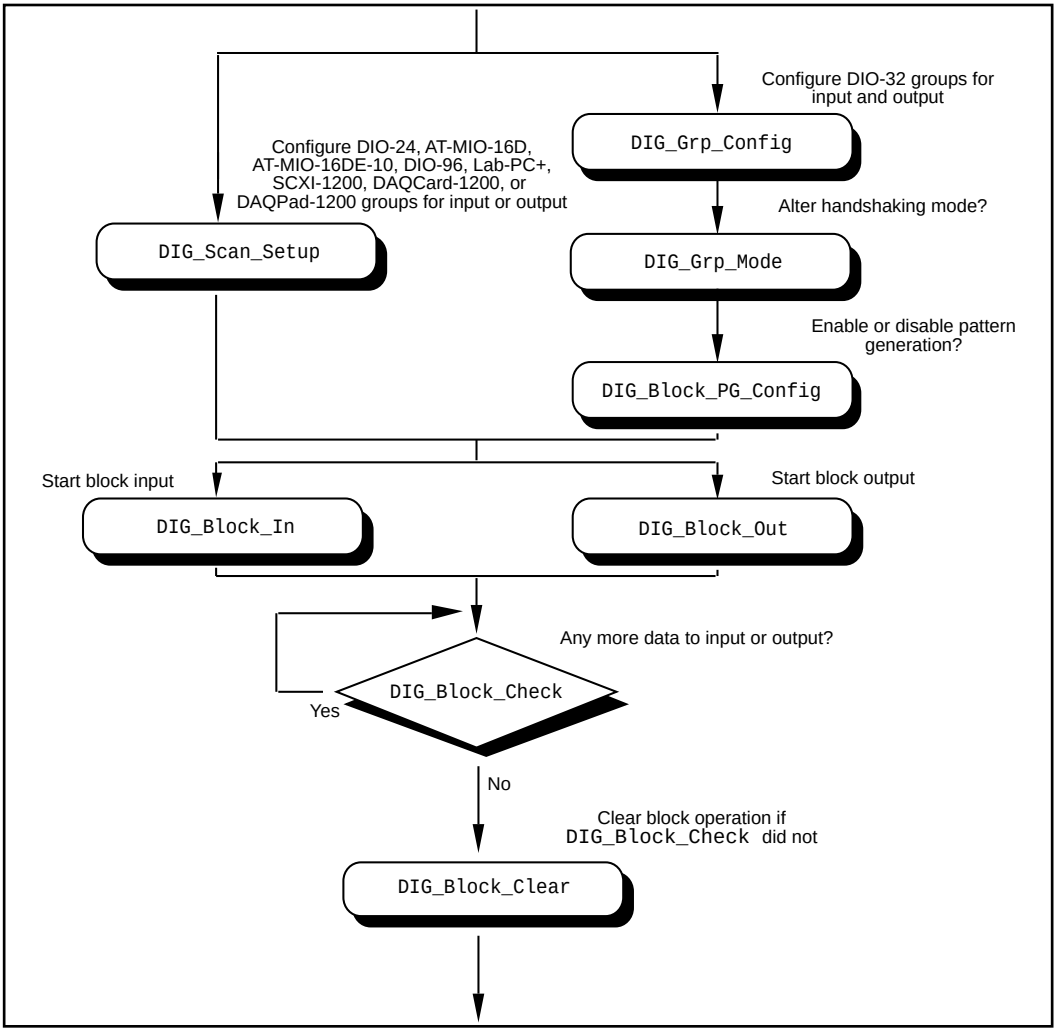


Figure 3-24. Digital Block Input or Output Application

☞ **Note:** ***The DIO-32F, DIO-24, AT-MIO-16D, AT-MIO-16DE-10, PC-DIO-96, Lab-PC+ , SCXI-1200, DAQPad-1200, and DAQCard-1200 can all***

perform group block operations. However, the DIO-24, AT-MIO-16D, AT-MIO-16DE-10, PC-DIO-96, Lab-PC+, SCXI-1200, DAQPad-1200, and DAQCard-1200 have special wiring requirements for groups larger than one port. The wiring for both the input and output cases for these devices is explained in the `DIG_SCAN_Setup` ***function description. No additional wiring is necessary for the DIO-32F.***

The first step for a group block I/O application is to call `DIG_Grp_Config` or `DIG_SCAN_Setup` to configure individual ports as a group. You should call `DIG_Grp_Config` if you have a DIO-32F. Call `DIG_SCAN_Setup` for all other devices. The AT-DIO-32F is restricted to group sizes of two and four ports for block I/O.

If you are using a DIO-32F, you can alter the handshaking mode of the group by calling `DIG_Grp_Mode`. Also, the DIO-32F can perform digital pattern generation. Pattern generation is simply reading in or writing out digital data at a fixed rate. This is the digital equivalent of analog waveform generation. To enable pattern generation, call `DIG_Block_PG_Config` as shown in Figure 3-24. You cannot handshake with pattern generation, so do not connect any handshaking lines. Refer to the explanation of pattern generation later in this chapter for more information.

The next step for your application, as illustrated in Figure 3-24, is to call `DIG_Block_In` or `DIG_Block_Out` to actually start the data transfer.

After you start the operation, you can call `DIG_Block_Check` to get the current progress of the transfer. If the block operation completes prior to a `DIG_Block_Check` call, `DIG_Block_Check` will automatically call `DIG_Block_Clear`, which performs cleanup work.

The final step of a digital block operation is to call `DIG_Block_Clear`. `DIG_Block_Clear` performs the necessary cleanup work after a digital block operation. You must call this function explicitly if `DIG_Block_Check` did not already call `DIG_Block_Clear`.

☞ **Note:** `DIG_Block_Clear` ***halts any ongoing block operation. Therefore, call*** `DIG_Block_Clear` ***only if you are certain the block operation has completed or you want to stop the current operation.***

Digital Double-Buffered Group Block I/O Applications

You can also configure group block operations as double-buffered operations for DIO-32 boards. With double-buffered operations, you can do continuous input or output with a limited amount of memory. See the *Double-Buffered I/O with the DIO-32F* section later in this chapter for an explanation of double buffering. Figure 3-25 outlines the basic steps for digital double-buffered group block I/O applications.

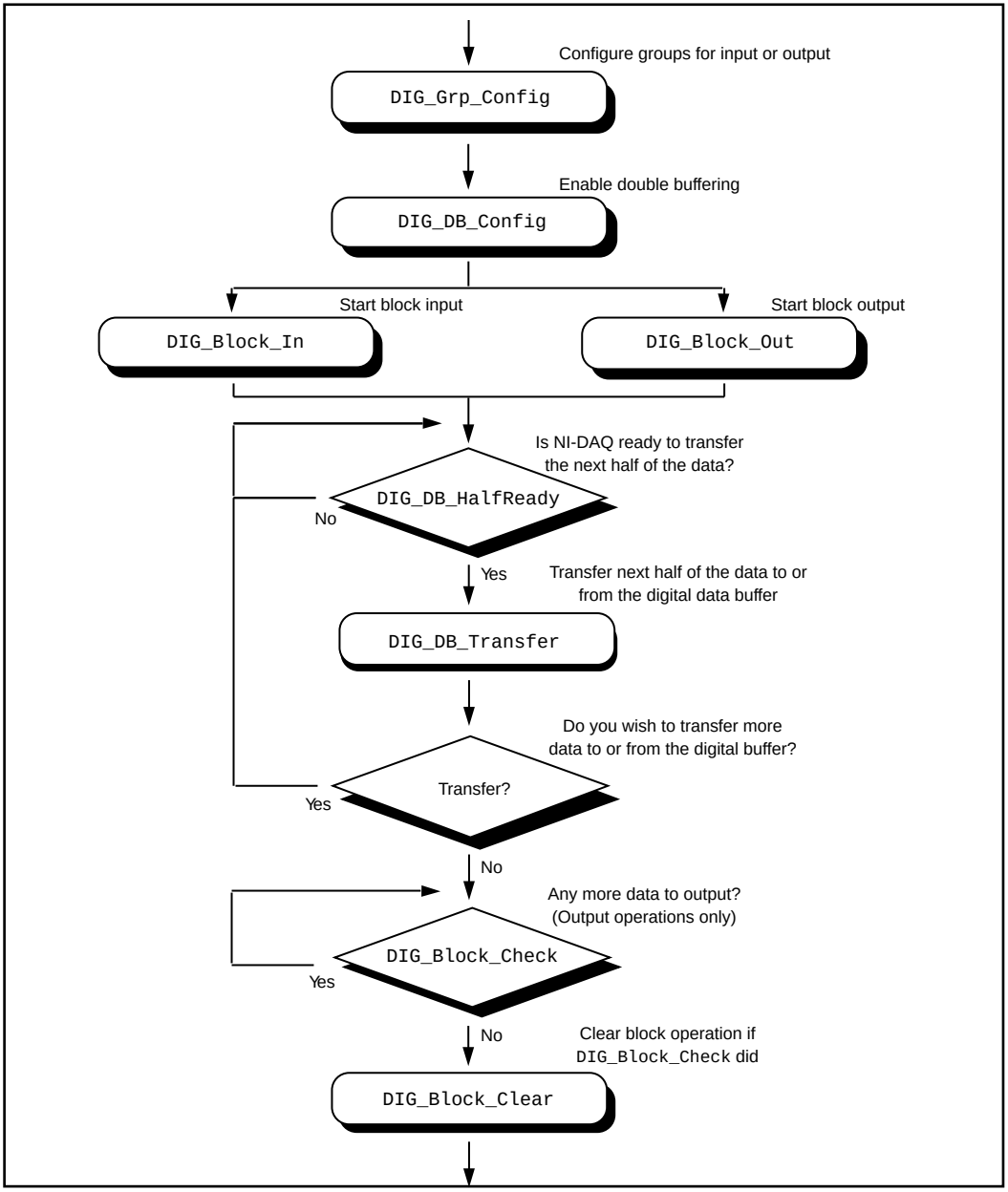


Figure 3-25. Double-Buffered Block Operation

The first step for an application is to call DIG_Grp_Config to configure individual ports as a group. Although the steps have been left out of the diagram, you can alter the handshaking mode and enable pattern generation as shown in Figure 3-24, and explained in the *Digital Group Block I/O Applications* section earlier in this chapter. Next, enable double buffering by calling DIG_DB_Config (second step of Figure 3-25). To start the digital block input or output, call DIG_Block_In or DIG_Block_Out.

After the operation has started, you may perform any number of transfers to or from the circular buffer. Input operations will transfer new data from the digital buffer for storage or processing. Output operations will transfer new data to the digital buffer for output.

To transfer to or from the circular buffer, call the DIG_DB_Transfer function. After you call the function, NI-DAQ waits until it can transfer the data before returning to the application. To avoid the waiting period, call DIG_DB_HalfReady to determine if NI-DAQ can make the transfer immediately. If DIG_DB_HalfReady indicates that NI-DAQ is not ready for a transfer, your application is free to do other processing and check the status later.

After the final transfer, you may want to call DIG_Block_Check to get the current progress of the transfer. For example, if you are using double-buffered output, NI-DAQ requires some time after the final transfer to actually output the data. In addition, if NI-DAQ completes the block operation prior to a DIG_Block_Check call, DIG_Block_Check will automatically call DIG_Block_Clear to perform cleanup work.

The final step of a double-buffered block operation is to call DIG_Block_Clear. DIG_Block_Clear performs the necessary cleanup work after a digital block operation. You must explicitly call this function if DIG_Block_Check did not already call it.

☞ **Note:** DIG_Block_Clear ***halts any ongoing block operation. Therefore, call*** DIG_Block_Clear ***only if you are certain the block operation has completed or if you want to stop the current operation.***

Pattern Generation I/O with the DIO-32F

For clocked digital I/O, where a port is written to or read from based on the output of a counter, use pattern generation. The

DIG_Block_PG_Config function enables the pattern generation mode of digital I/O. When pattern generation is enabled, a subsequent DIG_Block_In or DIG_Block_Out call automatically uses this mode. Each group has its own onboard counter and so each can simultaneously run in this mode at different rates. Also, you use an external counter by connecting its output to the appropriate REQ pin at the I/O connector. For an input group, pattern generation is analogous to a data acquisition operation, but instead of reading analog input channels, the digital ports are read. For an output group, pattern generation is analogous to waveform generation, but instead of writing voltages to the analog output channels, NI-DAQ writes digital patterns to the digital ports.

The DIO-32F uses DMA to service pattern generation. However, certain buffers require NI-DAQ to reprogram the DMA controller during the pattern generation. The extra time needed to do reprogramming increases the minimum request interval (thus decreasing the maximum rate). Chapter 4, *DMA and Programmed I/O Performance Limitations*, explains performance limitations in DOS and Windows for DMA operations. Use the utility function Align_DMA_Buffer to avoid the negative effects of page boundaries on PC AT and compatible computers when using pattern generation at update intervals faster than about 50 µs (this number will depend on your PC).

☞ **Note:** DIG_Block_In ***and*** DIG_Block_Out ***return a warning (*****15 DMAReprogramming*****) if reprogramming of the DMA controller is necessary. Also, page boundaries in a buffer that is to be used for 32-bit digital pattern generation causes unpredictable results for AT bus computer users, regardless of the request interval used.***

For AT bus computers, use the utility function Align_DMA_Buffer to avoid the negative effects of page boundaries in the following cases:

- When using digital I/O pattern generation at request intervals that are smaller than those listed in Table 5-2 for buffers with page boundaries.
- When using 32-bit digital I/O pattern generation at any speed.

To use Align_DMA_Buffer, however, you must allocate a buffer that is larger than the sample count to make room for Align_DMA_Buffer to move the data around. To guarantee that there is enough room for alignment, allocate twice as much space in your buffer (for example, if you have 5,000 samples, allocate space for

10,000 samples). When the buffer is aligned, make the normal calls to `DIG_Block_In` and `DIG_Block_Out`. A call to `DIG_Block_Clear` (either directly or indirectly through `DIG_Block_Check`) unaligns the data buffer if the data buffer was previously aligned by a call to `Align_DMA_Buffer`. To use the `Align_DMA_Buffer` utility function, follow these steps:

1. Allocate a buffer twice as large as the number of data samples you are generating.
2. If you are using digital output, build your digital pattern in the buffer.
3. Call `DIG_Grp_Config` for port assignment.
4. Call `DIG_Block_PG_Config` to enable pattern generation.
5. Call `Align_DMA_Buffer`, as described in Chapter 2, *Function Reference*, of the *NI-DAQ Function Reference Manual*.
6. Call `DIG_Block_In` or `DIG_Block_Out` with the aligned buffer to initiate the process.
7. Call `DIG_Block_Clear` after the pattern generation is finished.
8. Because `DIG_Block_Clear` unaligns the buffer, you can access the digital input pattern generation as you would with an unaligned buffer. If you want to use the same buffer again for digital output pattern generation, you must call `Align_DMA_Buffer` again.

Double-Buffered I/O with the DIO-32F

With the double-buffered (`DIG_DB`) digital I/O functions, you can input or output unlimited digital data without requiring unlimited memory. Double-buffered digital I/O is useful for applications such as streaming data to disk and sending long data streams as output to external devices. For an explanation of double-buffering, refer to Chapter 5, *NI-DAQ Double Buffering*.

Digital double-buffered output operations have two options. The first option is to stop the digital block operation if old data is ever encountered. This occurs if the `DIG_DB_Transfer` or `DIG_DB_StrTransfer` function calls are not keeping pace with the data input or output rate; that is, new data is not transferred to or from the circular buffer quickly enough. For digital input, this option prevents the loss of incoming data. For digital output, this option prevents erroneous data from being transferred to an external device. If the group is configured for handshaking, an old data stop is only a pause and a call to one of the transfer functions will resume the digital

operation. For pattern generation, an old data stop forces you to clear and restart the block operation.

The second option, available only to output groups, is the ability to transfer data that is less than half the circular buffer size to the circular buffer. This option is useful when long digital data streams are being output but the size of the data stream is not evenly divisible by the size of half of the circular buffer. This option imposes the restriction that the double-buffered digital block output is halted when a partial block of data has been output. This means that the data from the first call to `DIG_DB_Transfer` or `DIG_DB_StrTransfer` with a count less than half the circular buffer size is the last data output by the device.

Notice however, that enabling either of the double-buffered digital output options causes an artificial split in the digital block buffer, requiring DMA reprogramming at the end of each half buffer. For a group that is configured for handshaking, such a split means that a pause in data transfer can occur while NI-DAQ reprograms the DMA. For a group configured for pattern generation, this split can 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. Both options can be enabled or disabled by the `DIG_DB_Config` function.

☞ **Note:** ***EISA chaining is disabled if partial transfers are enabled.***

The Counter/Timer Function Group

The Counter/Timer function group contains three sets—the Counter/Timer (CTR) functions, which perform timing I/O and counter operations such as pulse generation, frequency generation, and event counting, the Interval Counter/Timer (ICTR) functions, which perform interval timing I/O and counter operations, and General-Purpose Counter/Timer (GPCTR) functions, which perform various counting and timing operations.

Device Support for the Counter/Timer and Interval Counter/Timer Functions

Use the Counter/Timer (CTR) functions with the following devices:

- Am9513-based devices
- EISA-A2000
- PC-TIO-10

Use the Interval Counter (ICTR) functions with the following devices:

- DAQCard-500
- DAQCard-700
- Lab-PC+
- PC-LPM-16
- SCXI-1200
- DAQPad-1200
- DAQCard-1200

Use the General-Purpose Counter/Timer (GPCTR) functions with the E Series devices. Please refer to the GPCTR functions in the *NI-DAQ Function Reference Manual for PC Compatibles* for a detailed description of how to use the GPCTR functions for a variety of applications.

The Counter/Timer Functions

The Counter/Timer (CTR) functions perform timing I/O and counter operations on the Am9513-based devices, PC-TIO-10, and EISA-A2000:

CTR_Config	Specifies the counting configuration to use for a counter.

`CTR_EvCount`	Configures the specified counter for an event-counting operation and starts the counter.
`CTR_EvRead`	Reads the current counter total without disturbing the counting process and returns the count and overflow conditions.
`CTR_FOUT_Config`	Disables or enables and sets the frequency of the 4-bit programmable frequency output.
`CTR_Period`	Configures the specified counter for period or pulse-width measurement.
`CTR_Pulse`	Causes the specified counter to generate a specified pulse-programmable delay and pulse width.
`CTR_Rate`	Converts frequency and duty-cycle values of a desired square wave into the timebase and period parameters needed for input to the `CTR_Square` function that produces the square wave.
`CTR_Reset`	Turns off the specified counter operation and places the counter output drivers in the selected output state.
`CTR_Restart`	Restarts operation of the specified counter.
`CTR_Simul_Op`	Configures and simultaneously starts and stops multiple counters.
`CTR_Square`	Causes the specified counter to generate a continuous square wave output of specified duty cycle and frequency.
`CTR_State`	Returns the OUT logic level of the specified counter.

`CTR_Stop`	Suspends operation of the specified counter so that NI-DAQ can restart the counter operation.

Counter/Timer Operation for the CTR Functions

Figure 3-26 shows the 16-bit counters available on the Am9513-based devices, PC-TIO-10, and EISA-A2000.

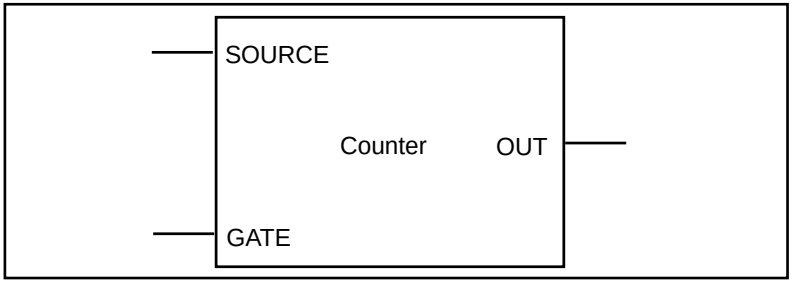


Figure 3-26. Counter Block Diagram

Each counter has a SOURCE input, a GATE input, and an output labeled OUT.

The counters can use several timebases for counting operations. A counter can use the signal supplied at any of the Am9513 five SOURCE or GATE inputs for counting operations. The Am9513 also makes available five internal timebases that any counter can use:

- 1 MHz clock (1 µs resolution)
- 100 kHz clock (10 µs resolution)
- 10 kHz clock (100 µs resolution)
- 1 kHz clock (1 ms resolution)
- 100 Hz clock (10 ms resolution)

☞ ***Note:*** ***A 5 MHz internal timebase (200 ns resolution) is also available on SOURCE 2 of the Am9513 on the AT-MIO-16F-5, AT-MIO-16X, and AT-MIO-64F-5, and on SOURCE 5 for counters 1 to 5 and SOURCE 10 for counters 6 to 10 on the PC-TIO-10.***

In addition, you can program the counter to use the output of the next lower-order counter as a signal source. This arrangement is useful for

counter concatenation. For example, you can program counter 2 to count the output of Counter 1, thus creating a 32-bit counter.

You can configure a counter to count either falling or rising edges of the selected internal timebase, SOURCE input, GATE input, or next lower-order counter signal.

You can use the counter GATE input to gate counting operations. After you configure a counter through software for an operation, a signal at the GATE input can start and stop the counter operation. There are nine gating modes available in the Am9513:

- No Gating—Counter is started and stopped by software.
- High-Level Gating—Counter is active when its gate input is at high-logic state. The counter is suspended when its gate input is at low-logic state.
- Low-Level Gating—Counter is active when its gate input is at low-logic state. The counter is suspended when its gate input is at high-logic state.
- Rising Edge Gating—Counter starts counting when it receives a low-to-high edge at its gate input.
- Falling Edge Gating—Counter starts counting when it receives a high-to-low edge at its gate input.
- High Terminal Count Gating—Counter is active when the next lower-order counter reaches terminal count (TC) and generates a TC pulse.
- High-Level Gate N+1 Gating—Counter is active when the gate input of the next higher-order counter is at high-logic state. Otherwise, the counter is suspended.
- High-Level Gate N-1 Gating—Counter is active when the gate input of the next lower-order counter is at high-logic state. Otherwise, the counter is suspended.
- Special Gating—The gate input selects the reload source but does not start counting. The counter uses the value stored in its internal Hold register when the gate input is high, and uses the value stored in its internal Load register when the gate input is low.

Counter operation starts and stops relative to the selected timebase. When a counter is configured for no gating, the counter starts at the first timebase/source edge (rising or falling, depending on the selection) after the software configures the counter. When a counter is configured for gating modes, gate signals take effect at the next

timebase/source edge. For example, if a counter is configured to count rising edges and to use the falling edge gating mode, the counter starts counting on the next rising edge after it receives a high-to-low edge on its GATE input. Thus, some time is spent synchronizing the GATE input with the timebase/source. This synchronization time creates a time lapse uncertainty from 0 to 1 timebase period between the application of the signal at the GATE input and the start of the counter operation.

The counter generates timing signals at its OUT output. If the counter is not operating, you can set its output to one of three states—high-impedance state, low-logic state, or high-logic state.

The counters generate two types of output signals during counter operation—TC pulse output and TC toggled output. A counter reaches TC when it counts up (to 65,535) or down (to 0) and rolls over. In many counter applications, the counter reloads from an internal register when it reaches TC. In TC pulse output mode, the counter generates a pulse during the cycle in which it reaches TC. In TC toggled output mode, the counter output changes state on the next source edge after reaching TC. In addition, you can configure the counters for positive logic output or negative (inverted) logic output. Figure 3-27 shows examples of the four types of output signals generated.

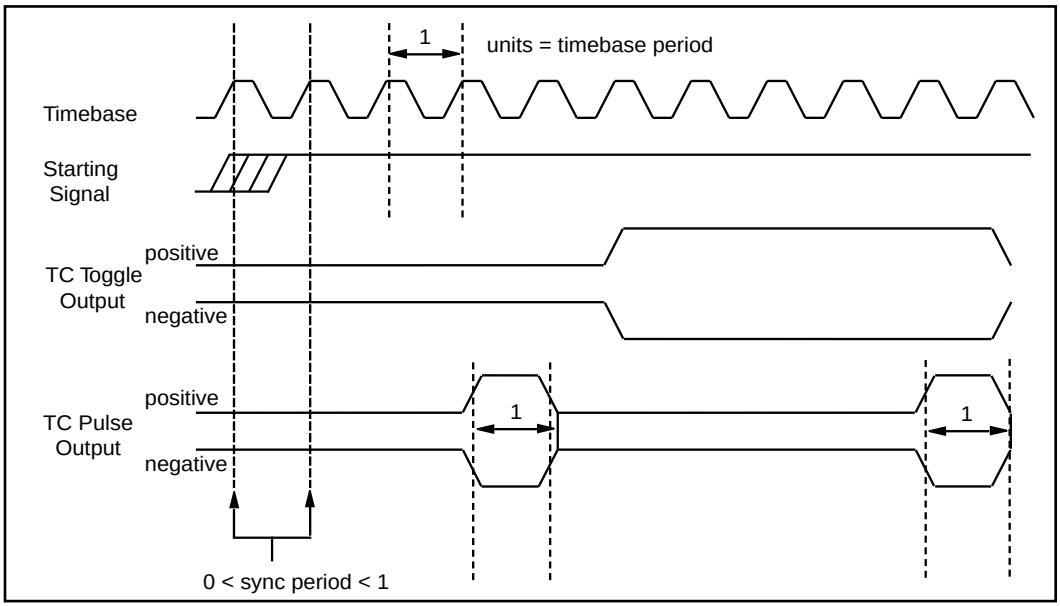


Figure 3-27. Counter Timing and Output Types

Figure 3-27 represents a counter generating a delayed pulse (see `CTR_Pulse`) and demonstrates the four forms the output pulse can take given the four different types of output signals supported. The TC toggled positive logic output looks like what would be expected when generating a pulse. For most of the Counter/Timer functions, TC toggled output is the preferred output configuration; however, the other signal types are also available. The starting signal, shown in Figure 3-27, represents either a software starting of the counter, for the no-gating case, or some sort of signal at the GATE input. The signal could be either a rising edge gate or a high-level gate. If the signal is a low-level or falling edge gate, the starting signal simply appears inverted. In Figure 3-27, the counter is configured to count the signal output changes state with respect to the rising edge of the timebase.

Programmable Frequency Output Operation

The Am9513-based devices and PC-TIO-10 provide a 4-bit programmable frequency output signal. This signal is a divided-down version of the selected timebase. Any of five internal timebases, counter SOURCE inputs, and counter GATE inputs can be selected as

the FOUT source. See the CTR_FOUT_Config function description in the *NI-DAQ Function Reference Manual* for FOUT use and timing information.

Counter/Timer Application Hints

All NI-DAQ counter/timer functions can be broken down into two major categories—event-counting functions and pulse generation functions. On the top of those functions, NI-DAQ has utility functions.

CTR_EvCount and CTR_EvRead are the two functions which are designed for event-counting. See Figure 3-28 for basic building blocks of event-counter applications. Also, read *Event-Counting Applications* later in this chapter for details.

Another major category of counter functions is pulse generation. With the NI-DAQ counter functions, you can call CTR_Pulse to generate a pulse or CTR_Square to generate a train of pulses (a square wave). To generate a pulse or a square wave, see Figure 3-29 for details on the function flow. When CTR_Square is used with 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 frequencies. If the output mode is TC Pulse, the result is two pulse trains of different frequencies.

Another type of gated pulse generation can be called *retriggerable one-shot pulse*, where a signal pulse is produced in response to a hardware trigger. To do this, call CTR_Config and specify edge gating. Connect your trigger signal to the GATE input. Call CTR_Square to specify your pulse. Subsequently, each edge sent to the GATE input will produce one cycle of the square wave.

Besides CTR_Square, you can also call CTR_FOUT_Config to generate a square wave. The advantage of using CTR_FOUT_Config is that it does not use a counter to generate the square wave. It uses a different built-in feature of the counter/timer chip. However, unlike CTR_Square, CTR_FOUT_Config can only generate a square wave with a 50-50 duty cycle.

NI-DAQ has a number of utility functions with which you have more control over the counters. CTR_State is for checking the logic level of any counter output. CTR_Reset halts any operation on a counter

and puts the counter output to a known state. `CTR_Stop` and `CTR_Restart` stop and restart any operation on a counter. `CTR_Simul_Op` can simultaneously start, stop, and restart any number of counters. Also, `CTR_Simul_Op` can simultaneously save all the current counter values to their hold registers, which you can read later, one at a time. See Figure 3-30 on how to incorporate `CTR_Simul_Op` with other counter functions like `CTR_EvCount` and `CTR_Pulse`.

Am9513-based device counter configuration settings applied through `CTR_Config` persist when waveform generation functions use the same counter. It is possible, for example, to configure the gating mode of a counter that will later be used for waveform generation so that you can externally trigger the operation with a pulse to the counter's gate. See the `CTR_Config` function in the *NI-DAQ Function Reference Manual for PC Compatibles* for more details.

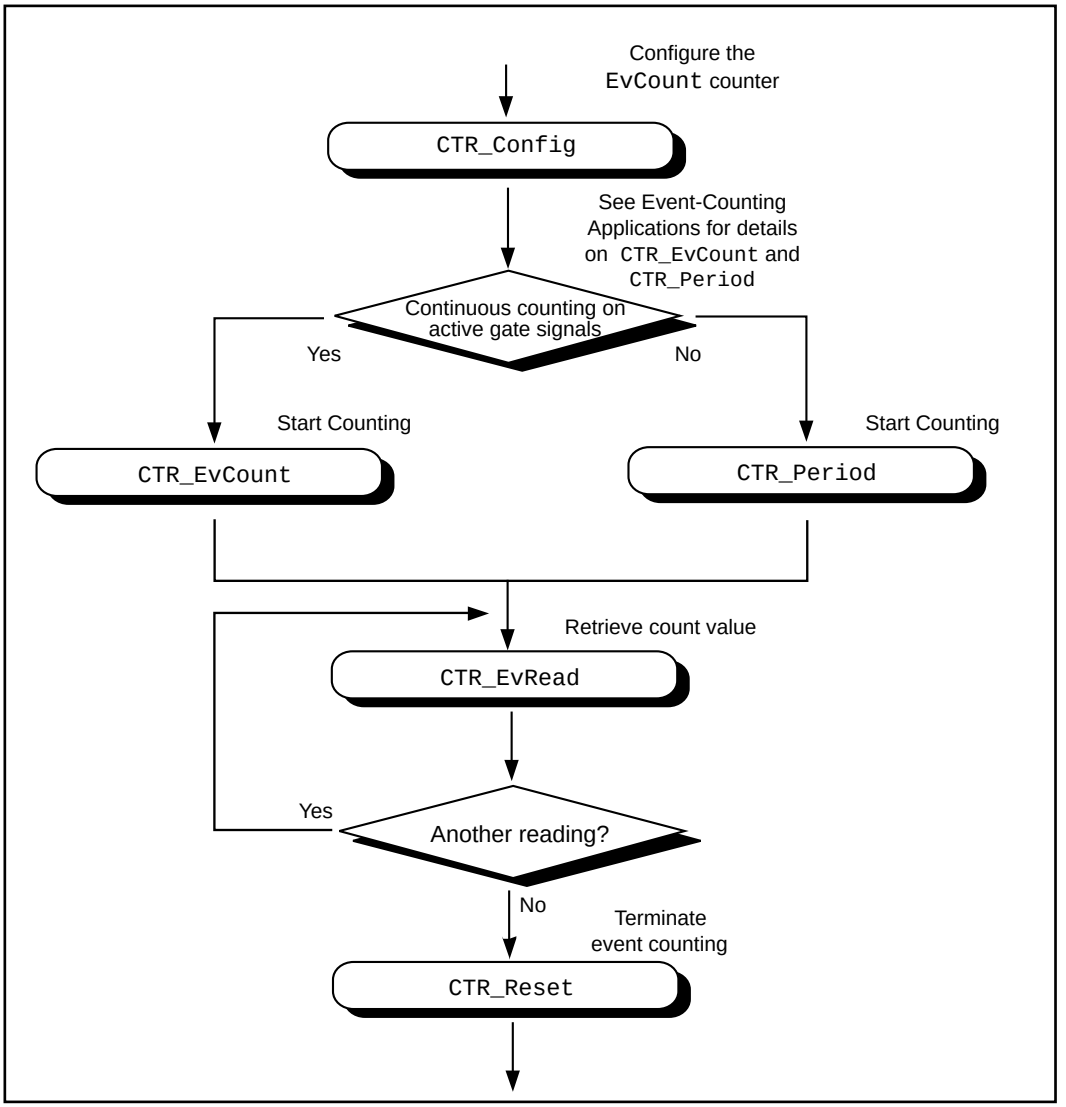


Figure 3-28. Event Counting

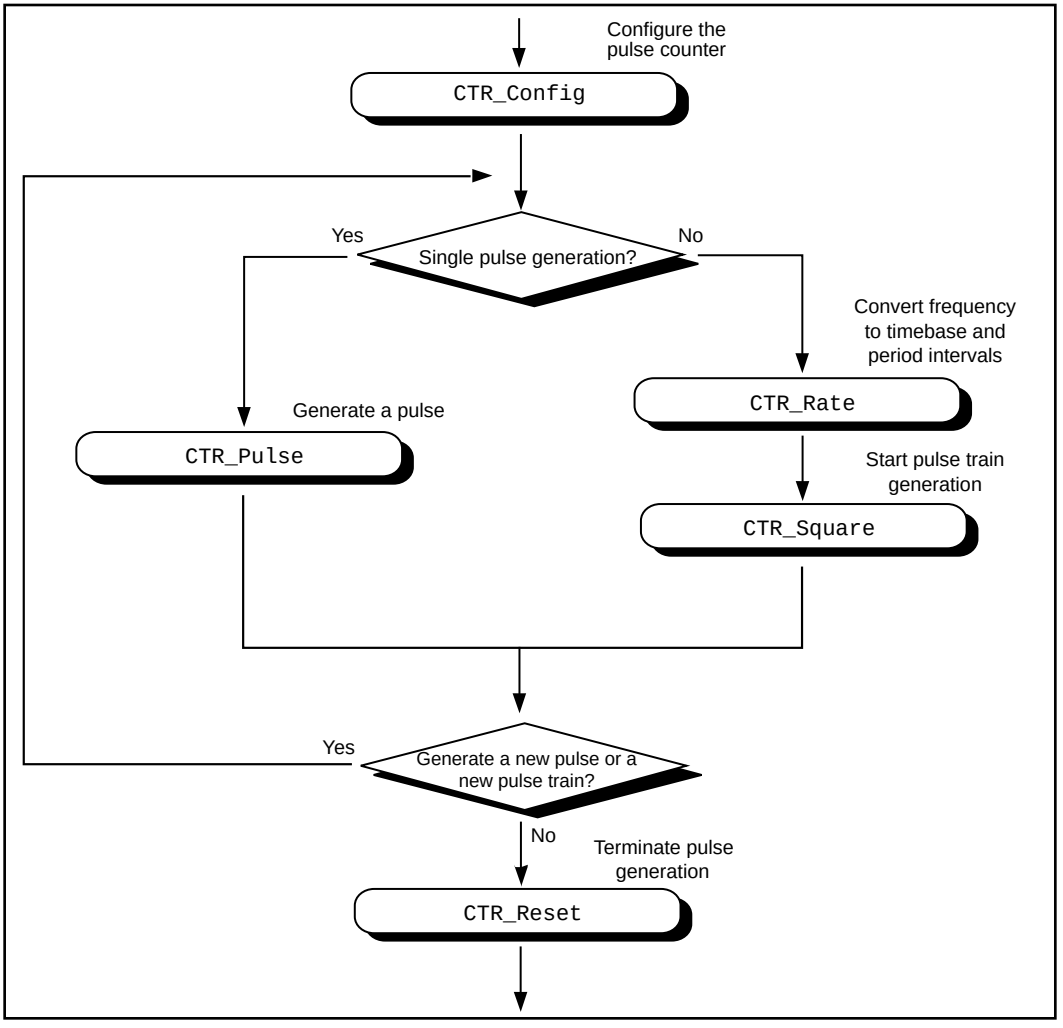


Figure 3-29. Pulse Generation

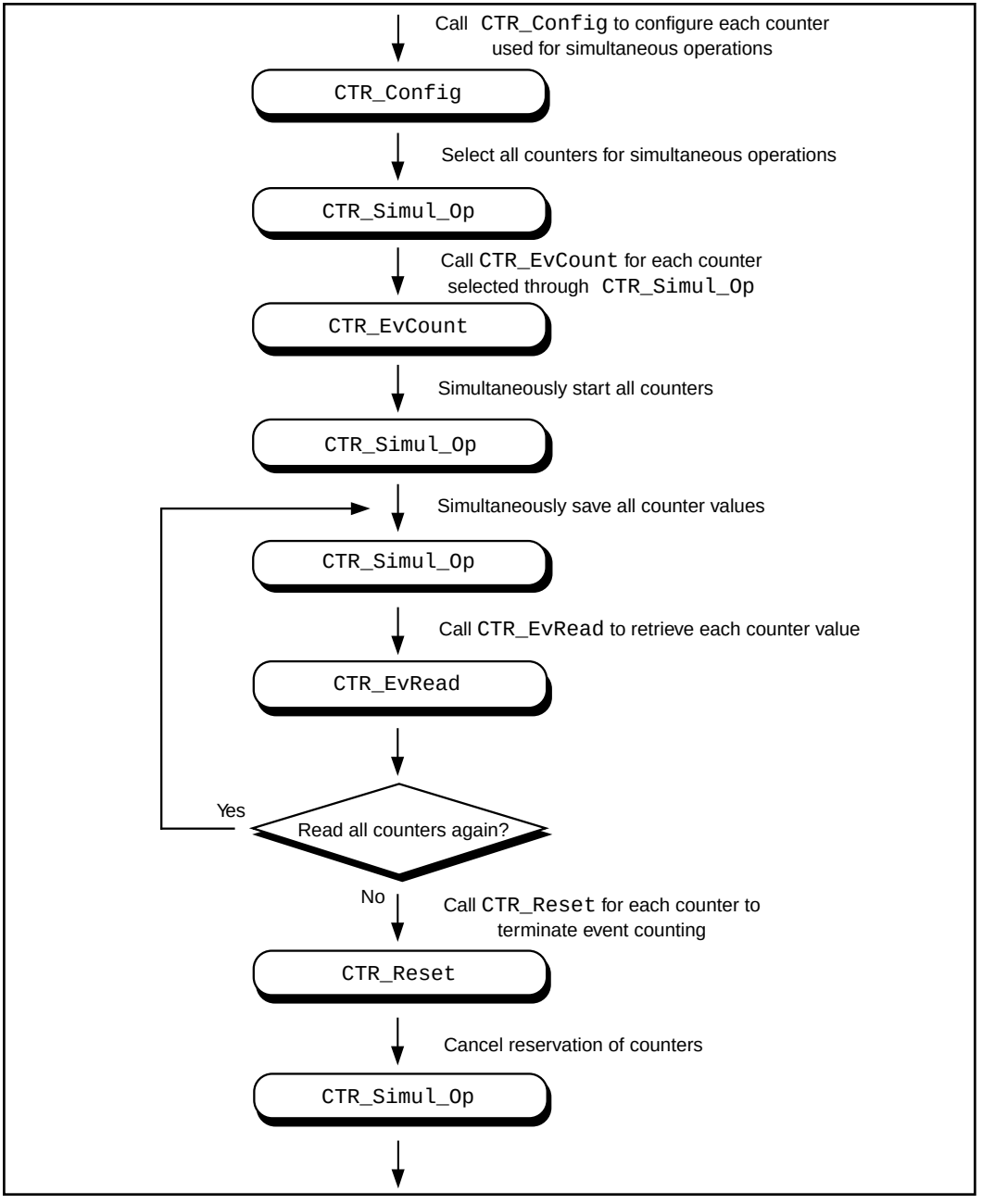


Figure 3-30. Simultaneous Counter Operation

Event-Counting Applications

`CTR_EvCount` and `CTR_EvRead` support four types of event-counting/timing measurements—event counting, pulse-width measurement, time-lapse measurement, and frequency measurement. `CTR_EvCount` also supports the concatenation of counters such that you can obtain 32-bit or 48-bit resolution for these measurements.

For event-counting applications, the events counted are the signal transitions or edges of an input SOURCE signal; therefore, you should set **timebase** to a value from 6 through 10. NI-DAQ can count either low-to-high or high-to-low edges (this feature is selected by **edgeMode** in the `CTR_Config` function). In addition, you can use the various gating modes of `CTR_Config` to control counting. Figure 3-31 illustrates timer event counting.

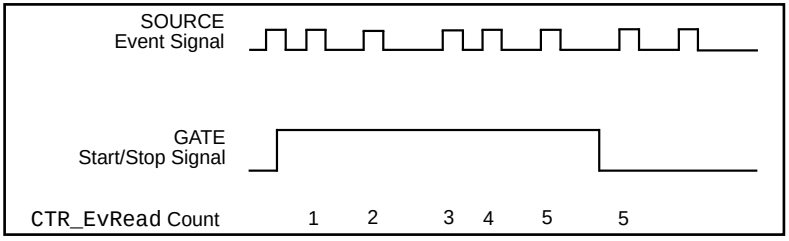


Figure 3-31. Timer Event Counting

For pulse-width measurement, you configure a counter to count for the duration of a pulse. For this application, you can use any timebase, including an external clock connected to the counter SOURCE input. Use level gating modes for pulse-width measurements in which the pulse to be measured is connected to the counter GATE input. Pulse width is then equal to (event count) ∗ (timebase period). Figure 3-32 shows a typical pulse-width measurement.

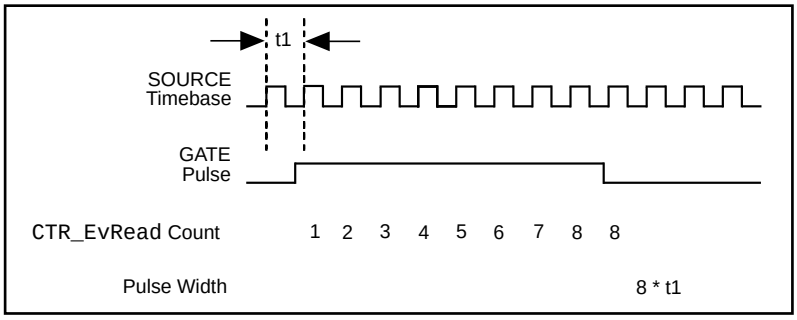


Figure 3-32. Pulse-Width Measurement

For time-lapse measurement, you configure a counter to count from the occurrence of some event. For this application, you can use any timebase, including an external clock connected to the counter SOURCE input. You can use edge-triggered gating modes if a single counter performs the event counting and if **cont** = 0. In this case, the starting event is an edge applied to the GATE input of the counter. The time lapse from the edge is then equal to (event count) ∗ (timebase period). If counters are to be concatenated for time-lapse measurement, use level gating where the GATE input signal goes active at the starting event and stays active.

Frequency measurement is a special case of event counting; that is, you can measure the frequency of an input signal by counting the number of edges of a signal that occur during a fixed amount of time. For this application, connect the signal to be measured to the SOURCE input of the counter and select the appropriate timebase (if **ctr** = 1, connect the signal to SOURCE1 and use **timebase** = 6). You can count either low-to-high or high-to-low edges (this feature is selected by **edgeMode** in the `CTR_Config` function). Using level gating and applying a gate pulse of a known, fixed duration to the GATE input of the counter constrains event counting to a fixed amount of time. The average frequency of the incoming signal is then equal to (event count)/(gate pulse width). Another MIO-16 counter can supply the gating pulse for frequency measurement by connecting the OUT signal from the counter producing the gating pulse to the GATE input of the counter doing the counting (see the `CTR_Pulse` function in the *NI-DAQ*

Function Reference Manual for more information). Figure 3-33 illustrates a frequency measurement.

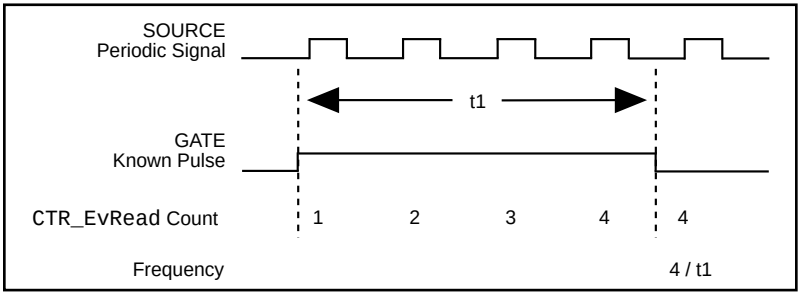


Figure 3-33. Frequency Measurement

For 16-bit resolution event counting and pulse-width, time-lapse, or frequency measurement, you need to use only one counter. Select **cont** = 0 so that you are notified if the counter overflows (see the `CTR_EvRead` function in the *NI-DAQ Function Reference Manual*). You can use any gating mode. In addition, select TC toggled output type and positive output polarity during the `CTR_Config` call so that overflow detection works properly.

For greater than 16-bit resolution, you can concatenate two or more counters. Configure a low-order counter to count the incoming edges or to measure the incoming pulse. Connect the OUT signal of the low-order counter to the SOURCE input of the next high-order counter by specifying a **timebase** of 0 for the next high-order counter. Configure the next high-order counter to count once every time the low-order counter rolls over. You can connect the OUT signal of the next high-order counter to the SOURCE input of an additional counter. The last counter (referred to as the high-order counter) is the counter that performs overflow detection. The lower-order counters increment continuously and generate output pulses whenever they roll over.

For 32-bit counting, use two counters. For 48-bit counting, use three counters, and so on. The counter configurations for concatenated event counting are as follows:

- Low-order counter configuration

  gateMode: either level gating or no gating

  edgeMode: any value

outType: TC pulse output type

outPolarity: positive polarity

timebase: any value

cont = 1: continuous counting

- Intermediate counter configuration

  edgeMode: count rising edges (indicates that the low-order counter rolled over)

  gateMode: no gating

  outType: TC pulse output type

  outPolarity: positive polarity

  timebase = 0: counts lower-order counter output

  cont = 1: continuous counting

- High-order counter configuration

  edgeMode: count rising edges (indicates that the low-order counter rolled over)

  gateMode: no gating

  outType: TC toggled output type (for proper overflow detection)

  outPolarity: positive polarity

  timebase = 0: counts lower-order counter output

  cont = 0: counter stops on overflow

Period and Continuous Pulse-Width Measurement Applications

With the proper use of `CTR_Config`, `CTR_Period`, and `CTR_EvRead`, you can configure a counter to make period or continuous pulse-width measurements.

To make a period measurement, call `CTR_Config` with **gateMode** set to either rising or falling edge-triggered gating (3 or 4). With rising edge-triggered gating, a counter can measure the time interval (t1 in Figure 3-34) between two rising edges of the gate signal. With falling edge-triggered gating, a counter can measure the time interval between two falling edges of the gate signal. After you call `CTR_Config` and apply the signal being measured to the appropriate gate, you can call `CTR_Period` to initiate period measurement. The specified counter starts counting on the first gate edge and latches the counter value to the onboard Hold Register after the counter detects a second gate edge.

After each period measurement, the counter reloads itself with a 0 and starts a new measurement. Figure 3-34 shows a continuous period measurement.

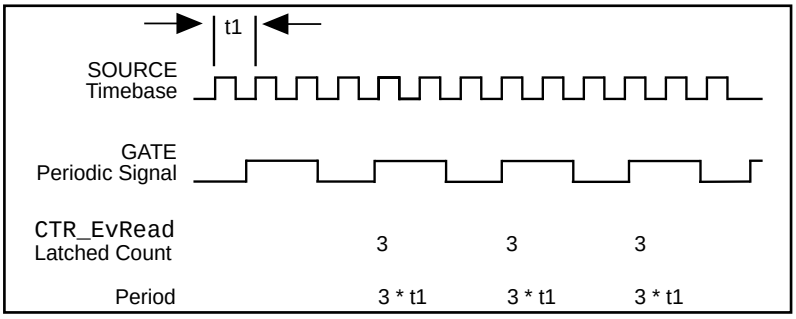


Figure 3-34. Continuous Period Measurement

While the measurement is occurring, call `CTR_EvRead` to retrieve the counter value saved in the Hold Register. The period is then equal to the value returned by `CTR_EvRead` ∗ **timebase**.

If you choose an improper timebase frequency, `CTR_EvRead` retrieves a smaller count value. A small count indicates that the timebase frequency is either too low or too high compared to the gate signal. If the timebase frequency is too low, the counter can only count a few source edges. However, if the timebase frequency is too high, the counter counts too many source edges, causing counter overflow. In case of counter overflow, a small count (typically 1 or 2) is saved on the Hold Register, and the counter reloads itself with a zero and waits for a new gate trigger to make a new measurement.

For a pulse-width measurement, use the same NI-DAQ calls used for period measurement, except that you should set **gateMode** to high-level or low-level gating (1 or 2). With high-level gating, a counter can measure the duration of a positive pulse. With low-level gating, a counter can measure the duration of a negative pulse. After you call `CTR_Period`, the counter starts counting after the gate becomes active. When the gate becomes inactive, the counter value latches to the Hold Register. You can then call `CTR_EvRead` to retrieve the saved value. Pulse width is then equal to the value returned by `CTR_EvRead` ∗ **timebase**. When the counter value is latched to the Hold Register, the counter reloads itself with a zero and waits for the gate to go active to begin a new measurement.

For measuring pulse-width, you need a rough estimate of the duration of the pulse being measured. When you configure a counter to measure pulse width, the counter continues counting in case of overflow. No counter value is latched to the Hold Register until the gate signal becomes inactive. To detect the counter overflow, feed the output of the pulse-width measurement counter to the source input of an event-counting counter. If the event-counting counter value is not zero after the pulse-width measurement, the pulse-width measurement is not correct.

The Interval Counter/Timer Functions

The Interval Counter/Timer functions perform interval timing I/O and counter operations on the DAQCard-500, DAQCard-700, Lab-PC+, SCXI-1200, DAQPad-1200, DAQCard-1200, and PC-LPM-16:

`ICTR_Read`	Returns the current contents of the selected counter without disturbing the counting process and returns the count.
`ICTR_Reset`	Sets the output of the selected counter to the specified state.
`ICTR_Setup`	Configures the given counter to operate in the specified mode.

Interval Counter/Timer Operation for the ICTR Functions

Figure 3-35 shows the 16-bit counters on the DAQCard-500, DAQCard-700, Lab-PC+, SCXI-1200, DAQPad-1200, DAQCard-1200, and PC-LPM-16.

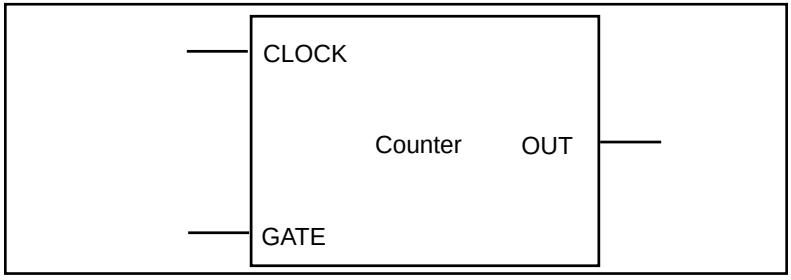


Figure 3-35. Interval Counter Block Diagram

Each counter has a clock input, a gate input, and an output. You can use a counter to count the falling edges of the signal applied to the CLK input. The counter gate input gates counting operations. Refer to the 8253 data sheet included in the *Lab-PC+ User Manual*, *SCXI-1200 Register-Level Programmer's Manual*, and *DAQPad-1200 Register-Level Programmer's Manual* and the MSM82C53 data sheet included in the *PC-LPM-16 User Manual, DAQCard-500 User Manual,* and *DAQCard-700 User Manual* to see how the gate inputs affect the counting operation in different counting modes.

Interval Counter/Timer Application Hints

NI-DAQ interval counter functions are an interface to the six different counting modes of 8253 counter chips on these devices. To choose the mode of operation, call `ICTR_Setup`. Refer to the `ICTR_Setup` function description in the *NI-DAQ Function Reference Manual* for descriptions of all six different counter modes.

After a counter is armed with `ICTR_Setup`, call `ICTR_Read` to retrieve the current counter value. Furthermore, to halt any counter operation, call `ICTR_Reset`.

The General-Purpose Counter/Timer Functions

The General-Purpose Counter/Timer (`GPCTR`) functions perform counting and timing operations on the E Series devices:

`GPCTR_Change_Parameter`	Customizes the counter operation to fit the requirements of your application by selecting a specific parameter setting.
`GPCTR_Configure_Buffer`	Assigns the buffer that NI-DAQ will use for a buffered counter operation.
`GPCTR_Control`	Controls the operation of the general-purpose counter.
`GPCTR_Set_Application`	Selects the application for which you will use the general-purpose counter. The function description in the *NI-DAQ Function Reference Manual* contains many application hints.

GPCTR_Watch	Monitors the state of the general-purpose counter and its operation.

The General-Purpose Counter/Timer Application Hints

The General-Purpose Counter/Timer (GPCTR) functions perform a variety of event counting, time measurement, and pulse and pulse train generation operations, including buffered operations. Refer to the description of the GPCTR_Set_Application function in the *NI-DAQ Function Reference Manual* for details associated with your application.

The Memory Management Functions

With the NI-DAQ Memory Management functions, you can access huge buffers (greater than 64 KB) that are not available in some programming environments such as BASIC and Pascal.

NI_DAQ_Mem_Alloc	Allocates a buffer from the system far heap, XMS memory, or DSP memory, and returns a handle to the buffer.
NI_DAQ_Mem_Attributes	Returns the attributes of the given memory object allocated through NI_DAQ_Mem_Alloc.
NI_DAQ_Mem_Copy	Copies data from or to memory allocated through NI_DAQ_Mem_Alloc.
NI_DAQ_Mem_Free	Releases the memory allocated through NI_DAQ_Mem_Alloc.
NI_DAQ_Mem_Lock	Locks a memory object and returns the lock handle of the memory object allocated through NI_DAQ_Mem_Alloc.
NI_DAQ_Mem_Unlock	Unlocks a memory object.

To allocate a buffer, call NI_DAQ_Mem_Alloc. When you are ready to begin a data acquisition process, call NI_DAQ_Mem_Lock with the handle returned by NI_DAQ_Mem_Alloc to lock your buffer and

pass the lock handle to the NI-DAQ function that initiates the data acquisition process. You can pass the lock handle to any NI-DAQ function that requires a buffer, such as `SCAN_Start`, `WFM_Load`, `DIG_Block_In`, and `MDAQ_Setup`.

To access the data in the buffer allocated through `NI_DAQ_Mem_Alloc`, call `NI_DAQ_Mem_Copy` to move the data from the buffer to an array to which you have direct access.

When you are done with the buffer, call `NI_DAQ_Mem_Unlock` and `NI_DAQ_Mem_Free`.

NI-DAQ has a standard header or include file for each supported compiler. For most applications, you should use these standard header files. However, if you will be using `NI_DAQ_Mem` arrays or DSP handles in the data acquisition, waveform generation, or digital functions, these standard header files will cause compiler warnings or compiler errors because most of the standard header files declare the buffer parameters in those functions as integer arrays (DSP handles and `NI_DAQ_Mem` arrays are long integers). You will receive the same warnings or errors if you are using float arrays in the data acquisition or waveform functions with an AT-DSP2200 board.

To eliminate the compiler warnings or errors, use the NI-DAQ header files with relaxed function prototypes if they are present for your compiler. However, when you use the relaxed prototypes, be more careful that you are passing the correct data types for all parameters in the NI-DAQ functions, and avoid passing numeric constants to these functions because the relaxed function prototypes may cause the compilers to do less data type checking and type conversion. If you pass the wrong data type to an NI-DAQ function, the function could return an error, or your application could fail.

For most compilers, you can typecast the `NI_DAQ_Mem` arrays or DSP handles when you pass them to NI-DAQ functions and use the standard header files instead of using the header files with relaxed prototypes. This way, the compilers still perform all of the data type checking and conversion for all of the function parameters.

If your compiler returns warnings because you pass `NI_DAQ_Mem` arrays or DSP handles to NI-DAQ functions, you can ignore the warnings.

The following list explains how different compilers react to the function prototypes in the header files, and what action you can take to alleviate problems. Remember, if you are not using NI_DAQ_Mem arrays or DSP handles, use the standard header files.

C compilers for DOS:

- Microsoft C
- Turbo C++
- Borland C++

The standard header file is NIDAQ.H. The Microsoft C compiler will return a warning that you can ignore if you pass an NI_DAQ_Mem array, a DSP handle, or a float array to an NI-DAQ function with the standard header file; the C++ compilers will return an error. You can do one of two things to eliminate the warnings or errors:

1. Use the NIDAQR.H header files that has relaxed function prototypes instead of the standard header file. Beware of the reduced data type checking and conversion.
2. You can typecast the DSP handle or NI_DAQ_Mem array to an integer pointer type when you call the NI-DAQ functions, as shown in the following example:

```
unsigned long bufHandle;
status = DAQ_Op (board, chan, gain, (int far *)bufHandle, count,
                 sampleRate);
```

C compilers for Windows:

- Microsoft C
- Borland Turbo C++

The standard header file for Microsoft C is WDAQ_C.H; for Borland C++ it is WDAQ_BC.H. The Microsoft compiler will return a warning that you can ignore if you pass a DSP handle or an NI_DAQ_Mem array to an NI-DAQ function; the C++ compiler will return an error. To eliminate the warning or error, you must typecast the DSP handle or NI_DAQ_Mem handle to an integer far pointer when you pass the handle to an NI-DAQ function as shown:

```
unsigned long bufHandle;
status = DAQ_Op (board, chan, gain, (int far *)bufHandle, count,
                 sampleRate);
```

Pascal compilers:

- Borland Turbo Pascal for DOS
- Turbo Pascal for Windows

For DOS, there is no header file; for Windows, the header file is `WDAQ_TP.INC`. You must typecast a DSP handle or an `NI_DAQ_Mem` array to a pointer when you pass it to an NI-DAQ function, as in the following example:

```
var bufHandle : Longint;
status := DAQ_Op (board, chan, gain, Pointer(bufHandle), count,
                  sampleRate);
```

Pascal for DOS users should also read the *Borland Turbo Pascal* section in Chapter 1, *Using the NI-DAQ Functions*, of the *NI-DAQ Function Reference Manual for PC Compatibles*.

Visual Basic for DOS:

The standard header file is `NIDAQ.INC`. The Visual Basic compiler will return errors if you pass a DSP handle or `NI_DAQ_Mem` array to the NI-DAQ functions using the standard header file. To eliminate the errors, use the `NIDAQR.INC` header file that has relaxed prototypes. Beware of the reduced data type checking and conversion.

Visual Basic for Windows:

The standard header file is `WDAQ_VB.BAS`, but if you use DSP handles or `NI_DAQ_Mem` arrays, use the `WDAQR_VB.BAS` header instead. `WDAQR_VB.BAS` has relaxed prototypes. Use the following calling convention:

```
status% = DAQ_Op (board%, chan%, gain%, BYVAL bufHandle&, count&,
                  sampleRate#)
```

The Visual Basic header files with relaxed prototypes do not cause the compilers to reduce the amount of data type checking and conversion on parameters other than those specified as relaxed.

The RTSI Bus Trigger Functions

The Real-Time System Integration (RTSI) Bus Trigger functions connect and disconnect signals over the RTSI bus trigger lines:

`RTSI_Clear`	Disconnects all RTSI bus trigger lines from signals on the specified device.
`RTSI_Clock`	Connects or disconnects the system clock from the RTSI bus.
`RTSI_Conn`	Connects a device signal to the specified RTSI bus trigger line.
`RTSI_DisConn`	Disconnects a device signal from the specified RTSI bus trigger line.

The following devices have an interface to the RTSI bus trigger lines:

- 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
- AT-MIO-16E-1
- AT-MIO-16E-2
- AT-MIO-16E-10
- AT-MIO-16DE-10
- AT-MIO-64E-4
- AT-MIO-16XE-50
- EISA-A2000
- NEC-AI-16E-4
- NEC-AI-16XE-50

- NEC-MIO-16E-4
- NEC-MIO-16XE-50

The RTSI Bus

The RTSI bus is implemented via a 34-pin ribbon cable connector on the AT Series and EISA-A2000 devices. Fourteen of the RTSI bus lines are dedicated to a seven-wire trigger bus. Each device that supports a RTSI bus interface contains a number of useful signals that can be driven onto, or received from, the trigger lines. Each device is equipped with a switch with which an onboard signal is connected to any one of the RTSI bus trigger lines through software control. By programming one device to drive a given trigger line and another device to receive from the same trigger line, you can hardware connect the two devices. You can use the RTSI Bus Trigger functions described in this chapter for this type of programmable signal interconnection between devices.

Through the RTSI bus, you can trigger one device from another device, share clocks and signals between devices, and synchronize devices to the same signals. The RTSI bus also can connect signals on a single device.

To specify the signals on each device that you can connect to the RTSI bus trigger lines, each device signal is assigned a signal code number. Make all references to that signal by using the signal code number in the RTSI Bus Trigger function calls. The signal codes for each device that can use the RTSI bus trigger lines are given later in this section.

Each signal listed in this chapter also has a signal direction. If a signal is listed with a source direction, that signal can drive the trigger lines. If a signal is listed with a receiver direction, that signal must be received from the trigger lines. A bidirectional signal direction means that the signal can act as either a source or a receiver, depending on the application.

MIO-16 and AT-MIO-16D RTSI Connections

The MIO-16 and AT-MIO-16D contain nine signals that you can connect to the RTSI bus trigger lines. Table 3-6 shows these signals.

Table 3-6. MIO-16/16D RTSI Bus Signals

Signal Name	Signal Direction	Signal Code
EXTCONV*	Bidirectional	0
FOUT	Source	1
OUT2	Source	2
GATE1	Receiver	3
SOURCE5	Receiver	4
OUT5	Source	5
STOP TRIG	Receiver	6
OUT1	Source	7
START TRIG*	Bidirectional	8

The signals GATE1, SOURCE5, OUT1, OUT2, OUT5, and FOUT are input and output signals from the Am9513 Counter/Timer on the MIO-16 board. OUT1, OUT2, and OUT5 are outputs of counters 1, 2, and 5, respectively. FOUT is the Am9513 programmable frequency output. GATE1 is the gating signal for counter 1, and SOURCE5 is a counter source input. The counters and frequency output are programmed via the counter functions (see *The Counter/Timer Functions* section earlier in this chapter for more information). GATE1, OUT1, OUT2, OUT5, and FOUT are also available on the device I/O connector.

The signals EXTCONV*, STOP TRIG, and START TRIG* are used for data acquisition timing. These signals are explained in the `DAQ_Config` and `DAQ_StopTrigger_Config` function descriptions in Chapter 2, *Function Descriptions*, in the *NI-DAQ Function Reference Manual for PC Compatibles*.

AT-MIO-16F-5, AT-MIO-64F-5, and AT-MIO-16X RTSI Connections

The AT-MIO-16F-5, AT-MIO-64F-5, and AT-MIO-16X contain nine signals that you can connect to the RTSI bus trigger lines. Table 3-7 shows these signals.

Table 3-7. AT-MIO-16F-5, AT-MIO-64F-5, and AT-MIO-16X RTSI Bus Signals

Signal Name	Signal Direction	Signal Code
EXTCONV*	Bidirectional	0
FOUT	Source	1
OUT2	Source	2
GATE1	Receiver	3
SOURCE5	Bidirectional	4
OUT5	Source	5
DACUPTRIG*	Receiver	6
OUT1	Bidirectional	7
EXTTRIG*	Bidirectional	8

The signals GATE1, SOURCE5, OUT1, OUT2, OUT5, and FOUT are input and output signals from the Am9513 Counter/Timer on the device. OUT1, OUT2, and OUT5 are outputs of counters 1, 2, and 5, respectively. FOUT is the Am9513 programmable frequency output. GATE1 is the gating signal for counter 1, and SOURCE5 is a counter source input. Program the counters and frequency output via the counter functions (see *The Counter/Timer Functions* section earlier in this chapter for more information). GATE1, OUT1, OUT2, OUT5, and FOUT are also available on the I/O connector of the AT-MIO-16F-5, AT-MIO-64F-5, and AT-MIO-16X.

Use the signals EXTCONV* and EXTTRIG* for data acquisition timing. These signals are explained in *The Data Acquisition Functions* section earlier in this chapter. The DACUPTRIG* signal and one of the OUT*x* signals (usually OUT5) are for waveform generation.

E Series Devices RTSI Connections

For information regarding signals on the E Series devices that you can connect to the RTSI bus, refer to the `Select_Signal` function description in Chapter 2, *Function Reference*, of the *NI-DAQ Function Reference Manual for PC Compatibles*.

AT-AO-6/10 RTSI Connections

The AT-AO-6/10 contains six signals that you can connect to the RTSI bus trigger lines. Table 3-8 shows these signals.

Table 3-8. MIO-16/16D RTSI Bus Signals

Signal Name	Signal Direction	Signal Code
OUT0*	Source	0
GATE2	Receiver	1
EXTUPD*	Source	2
OUT2*	Source	3
OUT1*	Source	4
EXTUPDATE*	Bidirectional	5

The signals GATE2, OUT0*, OUT1*, and OUT2* are input and output signals from the MSM82C53 Counter/Timer on the AT-AO-6/10 board. OUT0*, OUT1*, and OUT2* are outputs of counters 0, 1, and 2, respectively. GATE2 is the gating signal for counter 2.

The signals EXTUPDATE* and EXTUPD* externally update selected DACs. The EXTUPDATE* signal is shared with the I/O connector. For more information about the AT-AO-6/10 signals, see the *AT-AO-6/10 User Manual*.

DIO-32F RTSI Connections

The DIO-32F contains four signals that you can connect to the RTSI bus trigger lines. Table 3-9 shows these signals.

Table 3-9. DIO-32F RTSI Bus Signals

Signal Name	Signal Direction	Signal Code
REQ1	Receiver	0
REQ2	Receiver	1
ACK1	Source	2
ACK2	Source	3

The signals REQ1 and REQ2 are request signals received from the I/O connector. An external device drives these signals during handshaking. ACK1 and ACK2 are supplied for handshaking with the DIO-32F over the RTSI bus. For more information about the DIO-32F signals, see the *AT-DIO-32F User Manual*.

EISA-A2000 RTSI Connections

The EISA-A2000 contains seven signals that you can connect to the RTSI bus trigger lines. Table 3-10 shows these signals.

Table 3-10. EISA-A2000 RTSI Bus Signals

Signal Name	Signal Direction	Signal Code
START*	Bidirectional	0
TRIGGER*	Bidirectional	1
CLOCKO	Source	2
CLOCKI	Receiver	3
GATE2	Receiver	4
SOURCE2	Receiver	5
OUT2	Source	6

The signals GATE2, SOURCE2, and OUT2 are input and output signals from the Am9513 Counter/Timer on the EISA-A2000 board. GATE2 is the gating signal for counter 2, SOURCE2 is the source signal for counter 2, and OUT2 is the output of counter 2. Program counter 2 via the counter/Timer functions (see *The Counter/Timer Functions* section earlier in this chapter).

The signals START*, TRIGGER*, CLOCKO, and CLOCKI are for data acquisition timing. You can generate these signals locally on the EISA-A2000 or control them from the RTSI bus, as shown in the following table.

Signal Name	Description
START*	START* is an active-low signal that initiates a data acquisition sequence. If data acquisition is locally controlled, the START* signal pulses low when a trigger is generated from software or from the EISA-A2000 analog trigger or digital trigger circuitry (posttrigger or pretrigger mode). When locally generated, START* is a signal source. Alternatively, if the EISA-A2000 is configured to receive START* from the RTSI bus, a data acquisition sequence is initiated when the board receives a low pulse.
TRIGGER*	TRIGGER* is an active-low signal that activates the sample counter. If data acquisition is locally controlled, TRIGGER* pulses low when the board receives a trigger either through software (posttrigger mode only) or from the EISA-A2000 analog trigger or digital trigger circuitry (posttrigger or pretrigger mode). In pretrigger mode, TRIGGER pulses sometime after START*. In posttrigger mode, START* drives the TRIGGER* signal directly. The RTSI switch may drive the TRIGGER* signal when the EISA-A2000 is configured for pretrigger mode only.
CLOCKO	CLOCKO is the active-high, sample-clock output signal. The rising edge of this signal initiates a scanning sequence in which all active channels are simultaneously sampled. Any locally generated sample clock or any clock received from the I/O connector SAMPCLK* signal can drive CLOCKO.
CLOCKI	CLOCKI is the active-high, sample-clock input signal. If the RTSI switch drives CLOCKI, the rising edge of this signal initiates a scanning sequence in which all active channels are simultaneously sampled. The locally generated sample clock and I/O connector SAMPCLK* signal are ignored when the RTSI switch drives CLOCKI. The EISA-A2000 must use an external sample clock if the RTSI switch drives CLOCKI (see the `A2000_Config` function in the *NI-DAQ Function Reference Manual*).

☞ ***Note:*** ***If the RTSI switch drives any of the START*, TRIGGER*, or CLOCKI signals, locally generated signals are overwritten. The RTSI switch should drive TRIGGER* only if the EISA-A2000 is configured for pretrigger mode.***

AT-A2150 RTSI Connections

The AT-A2150 contains five signals that you can connect to the RTSI bus trigger lines. Table 3-11 shows these signals.

Table 3-11. AT-A2150 RTSI Bus Signals

Signal Name	Signal Direction	Signal Code
HWTrig*	Bidirectional	0
WCAD	Source	1
RTSITrig*	Source	2
SWTrig*	Source	3
RTSI_SWTrig*	Receiver	4

The signal WCAD is the conversion pulse signal that clocks the ADC. The signal RTSITrig* is a signal generated by writing to the RTSI Trigger Register on the AT-A2150 board. Currently, NI-DAQ does not have this functionality. You must explicitly write to the RTSI Trigger Register.

You can use the signals HWTrig*, SWTrig*, RTSI_SWTrig* for data acquisition timing. These signals may be generated locally or controlled from the RTSI bus. These signals are explained as follows:

Signal Name	Description
HWTrig*	HWTrig* is the digital trigger signal. When configured as a source, it can transmit the digital signal from the external I/O connector or the signal generated by the internal level-and-slope trigger circuit. When used as a receiver, the AT-A2150 can act on this signal as the trigger when configured for pretrigger, posttrigger, or posttrigger with delay mode. Only one board (the master) should have this line configured as a source. All other boards using this signal should be slaves.
SWTrig*	SWTrig* is a signal generated when a pretrigger, or software posttrigger acquisition has started. This can be connected to RTSI_SWTrig* on another AT-A2150 to initiate simultaneous acquisitions.
RTSI_SWTrig*	When RTSI_SWTrig* receives a signal, it has the same effect as writing to the A/D FIFO Start Register on the AT-A2150. This signal can start a pretrigger or software posttrigger acquisition.

When synchronizing multiple AT-A2150s, take the following steps:

1. Call `Master_Slave_Config` to specify the relationship.
2. If all AT-A2150s use a common timebase, call `RTSI_Clock` to synchronize the ADC clock signals.
3. Connect trigger signals from the master to the slave.

- For pretrigger mode, connect HWTrig* from the master to HWTrig* of the slave(s). Connect SWTrig* from the master to RTSI_SWTrig* of the slave(s).
- For posttrigger mode or posttrigger with delay mode, connect HWTrig* from the master to HWTrig* of the slave(s).
- For software posttrigger mode, connect SWTrig* from the master to RTSI_SWTrig* of the slave(s).

To remove synchronization of multiple AT-A2150s, use the following steps:

1. Call `RTSI_DisConn` to disconnect trigger lines.
2. Call `Master_Slave_Config` to remove the relationship.

3. If you used a common timebase, call `RTSI_Clock` to disconnect the ADC clock signals.

AT-DSP2200 RTSI Connections

The AT-DSP2200 has three signals that you can connect to the RTSI bus trigger lines. Table 3-12 shows these signals.

Table 3-12. AT-DSP2200 RTSI Bus Signals

Signal Name	Signal Direction	Signal Code
HWTrig*	Bidirectional	0
WCAD	Source	1
RTSITrig*	Source	2

The signal HWTrig* is the digital trigger signal. The AT-DSP2200 can receive this signal from the RTSI bus and use the signal as a trigger, or the board can use its internal level-and-slope trigger circuit as a trigger and also send this trigger to other boards via the RTSI bus. WCAD is the conversion pulse signal that clocks the ADC. RTSITrig* is a signal generated by writing to the RTSI Trigger Register on the AT-DSP2200. Currently, NI-DAQ does not have this functionality. You must explicitly write to the RTSI Trigger Register (with an output call in C, for example).

When synchronizing multiple AT-DSP2200s, perform the following steps:

1. Call `Master_Slave_Config` to specify the relationship.
2. If all AT-DSP2200s use a common timebase, call `RTSI_Clock` to synchronize the ADC clock signals.

To remove synchronization of multiple AT-DSP2200s, perform the following steps:

1. Call `Master_Slave_Config` to remove the relationship.
2. If you used a common timebase, call `RTSI_Clock` to disconnect the ADC clock signals.

☞ ***Note:*** ***You can synchronize multiple AT-DSP2200s and AT-A2150s with the previous steps.***

RTSI Bus Application Hints

This section gives a basic explanation of how to construct an application that uses RTSI bus NI-DAQ functions. Flowcharts are a quick reference for constructing potential applications from the NI-DAQ function calls.

An application that uses the RTSI bus has three basic steps. The first step is to connect the signals from the device to the RTSI bus. The next step is to actually execute the work of the application. The final step is to disconnect the signals from the RTSI bus. Figure 3-36 illustrates the normal order of RTSI function calls.

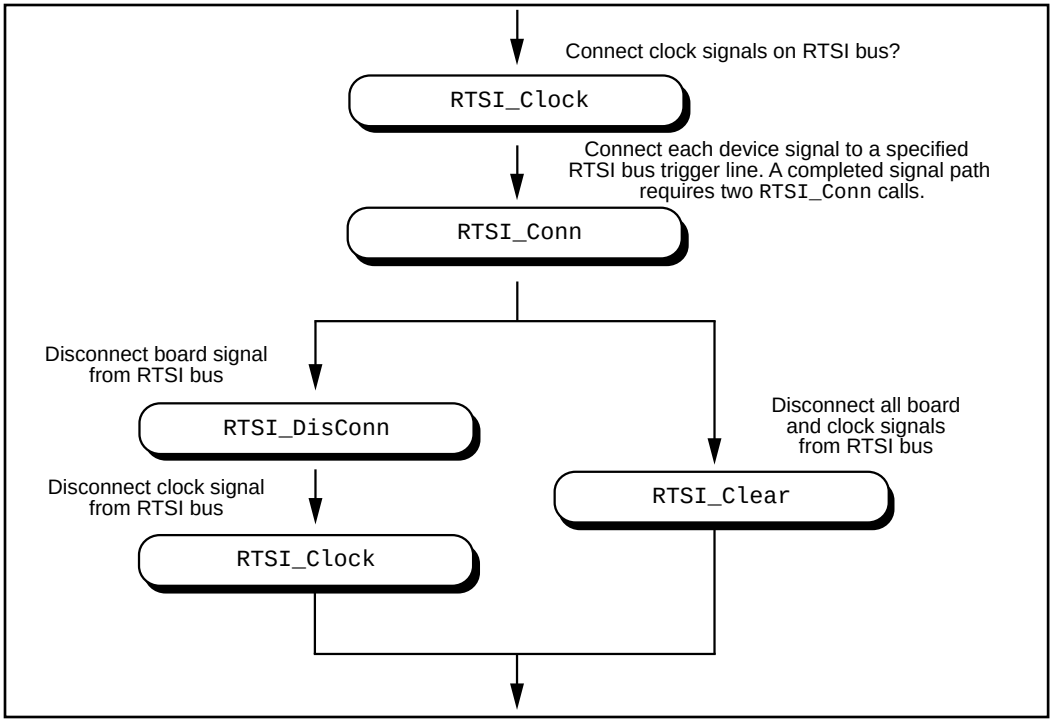


Figure 3-36. Basic RTSI Application Calls

Call `RTSI_Clock` and/or `RTSI_Conn` to connect the signals. Each completed signal path requires `RTSI_Conn` calls. The first call specifies the device signal to transmit onto a RTSI bus trigger line. The second call specifies the device signal that will receive a RTSI bus

trigger line. After the signals are connected, you are ready to do the actual work of your application.

After you are finished with the RTSI bus, disconnect the device from the bus. To do this, call `RTSI_DisConn` and or `RTSI_Clock` for each connection made. Alternatively, call `RTSI_Clear` to sever all connections from your device to the RTSI bus.

The SCXI Functions

`SCXI_AO_Write`	Sets the DAC channel on the SCXI-1124 to the specified voltage or current range. 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.
`SCXI_Cal_Constants`	Calculates calibration constants for the given channel and range or gain using measured voltage/binary pairs. You can use this function 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`.
`SCXI_Calibrate_Setup`	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 module or the

	SCXI-1121 module with the SCXI-1321 terminal block.
`SCXI_Change_Chan`	Selects a new channel of a multiplexed module that has previously been set up for a single-channel operation using the `SCXI_Single_Chan_Setup` function.
`SCXI_Configure_Filter`	Sets the specified channel to the given filter setting on any SCXI module that supports programmable filter settings (SCXI-1122 and SCXI-1141).
`SCXI_Get_Chassis_Info`	Returns chassis configuration information.
`SCXI_Get_Module_Info`	Returns configuration information for the given SCXI chassis slot number.
`SCXI_Get_State`	Gets the state of a single channel or an entire port on any digital or relay module.
`SCXI_Get_Status`	Reads the data in the status register on the specified module. You can use this function with the SCXI-1160 or SCXI-1122 to determine if the relays have finished switching, with the SCXI-1124 to determine if the DACs have settled, or with the SCXI-1102 to determine if the module has settled after changing gains.
`SCXI_Load_Config`	Loads the SCXI chassis configuration information that you established in the configuration utility. Sets the software states of the chassis and modules present to their default states. No changes are made to the hardware states of the SCXI chassis or modules.
`SCXI_MuxCtr_Setup`	Enables or disables the mux-counter during SCXI channel scanning to synchronize the MIO and AI device

	scan list with the module scan list that you have downloaded to Slot 0 of the SCXI chassis using the `SCXI_SCAN_Setup` call (MIO and AI devices only).
`SCXI_Reset`	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.
`SCXI_Scale`	Scales an array of binary data acquired from an SCXI channel to voltage.
`SCXI_SCAN_Setup`	Sets up the SCXI chassis for a multiplexed scanning data acquisition operation to be performed by the given DAQ device. The function downloads a module scan list to Slot 0 that will determine the sequence of modules that will be scanned and how many channels on each module will be scanned. Each module will be programmed with its given start channel. Any contention on the SCXIbus is resolved.
`SCXI_Set_Config`	Changes the configuration of the SCXI chassis that you established in the configuration utility. Sets the software states of the chassis and modules specified to their default states. No changes are made to the hardware states of the SCXI chassis or modules.
`SCXI_Set_Gain`	Sets the specified channel to the given filter setting on any SCXI module that supports programmable gain settings (SCXI 1100, SCXI-1102, SCXI-1122, and SCXI-1141).
`SCXI_Set_Input_Mode`	Configures the SCXI-1122 for 2-wire mode or 4-wire mode.

`SCXI_Set_State`	Sets the state of a single channel or an entire port on any digital or relay module.
`SCXI_Single_Chan_Setup`	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. You can also use this function to read the temperature sensor on a terminal block connected to the front connector of the module.
`SCXI_Track_Hold_Control`	Controls the Track/Hold state of an SCXI-1140 module that you have configured for a single-channel operation.
`SCXI_Track_Hold_Setup`	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.

SCXI Application Hints

There are three categories of SCXI applications—analog input applications, analog output applications, and digital applications.

Figure 3-37 shows the basic structure of an SCXI application.

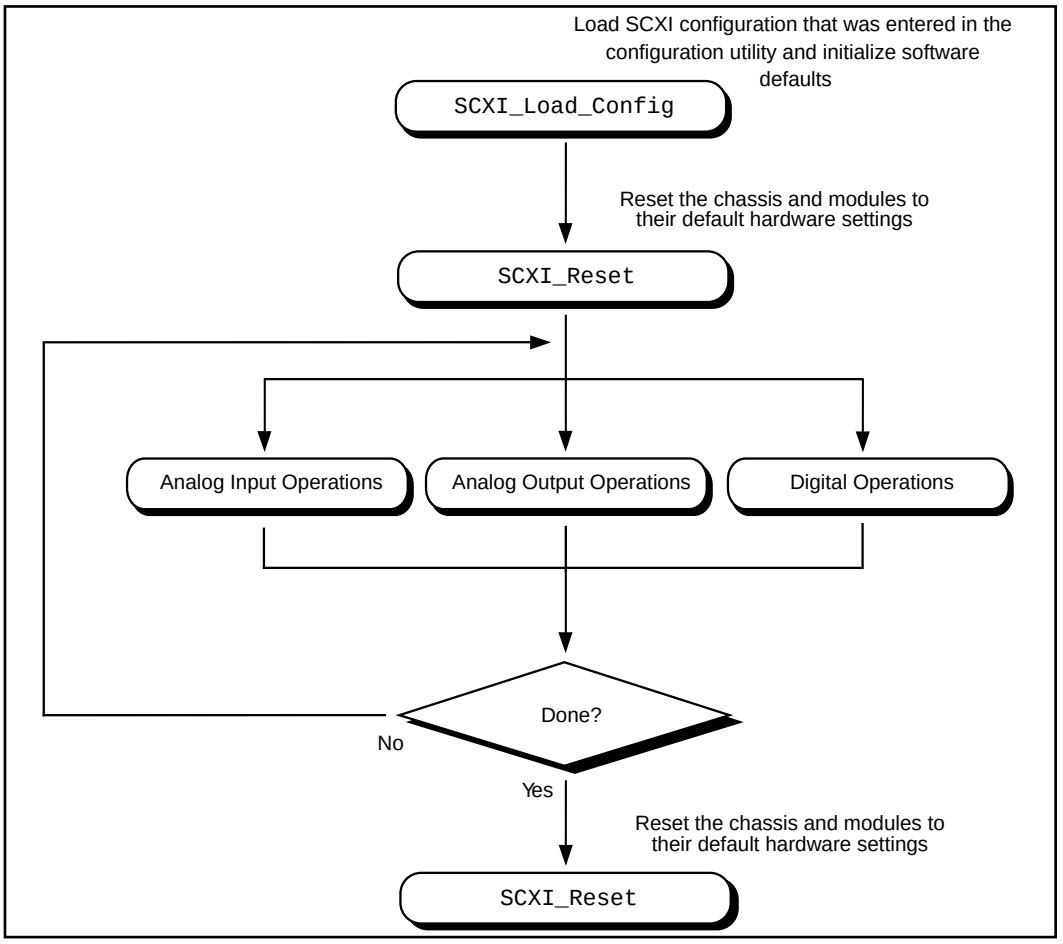


Figure 3-37. General SCXIbus Application

The figures in the following sections show the detailed call sequences for different types of SCXI operations. In effect, each of the remaining flowcharts in this section is an enlargement of the Analog Input Operations, the Analog Output Operations, or the Digital Operations node in Figure 3-37. Please refer to the function descriptions in Chapter 2, *Function Reference*, of the *NI-DAQ Function Reference Manual* for detailed information about each function used in the flowcharts.

The SCXI analog input applications can be divided further into two categories—single-channel applications and channel-scanning applications. The distinction between the two categories is simple—single-channel applications do not involve automatic channel switching by the hardware during an analog input process; channel-scanning applications do.

Single-channel applications use the AI or the DAQ class of functions described earlier in this chapter to acquire the input data after you have set up the SCXI system. To acquire data from more than one channel, you need multiple AI or DAQ function calls, and you may need explicit SCXI function calls to change the SCXI channel that has been selected; this specific type of single-channel application is referred to as *software scanning*.

Channel-scanning applications use the SCAN and Lab_ISCAN classes of functions described earlier in this chapter to acquire the input data after you have set up the SCXI system.

Building Analog Input Applications in Multiplexed Mode

Multiplexed applications require the use of SCXI functions to select the multiplexed channels, select the programmable module features, route signals on the SCXIbus, and program Slot 0. After you have set up the SCXI chassis and modules, you can use the AI, DAQ, SCAN, and Lab_ISCAN functions to acquire the data either with a plug-in DAQ device or the SCXI-1200. The **channel** parameter that is passed to each of these functions is almost always 0 because, by default, the multiplexed output of a module is connected to analog input channel 0 of the DAQ device or SCXI-1200. When you use multiple chassis, the modules in each chassis are multiplexed to a separate analog input channel. In that case, the **channel** parameters of the AI, DAQ, SCAN, and Lab_ISCAN functions should be the DAQ device channel that corresponds to the desired chassis for the operation. You cannot use the SCXI-1200 with multiple chassis.

Figure 3-38 shows the function call sequence of a single-channel or software-scanning application using an SCXI-1100, SCXI-1102, SCXI-1120, SCXI-1121, SCXI-1122, or SCXI-1141 module operating in Multiplexed mode.

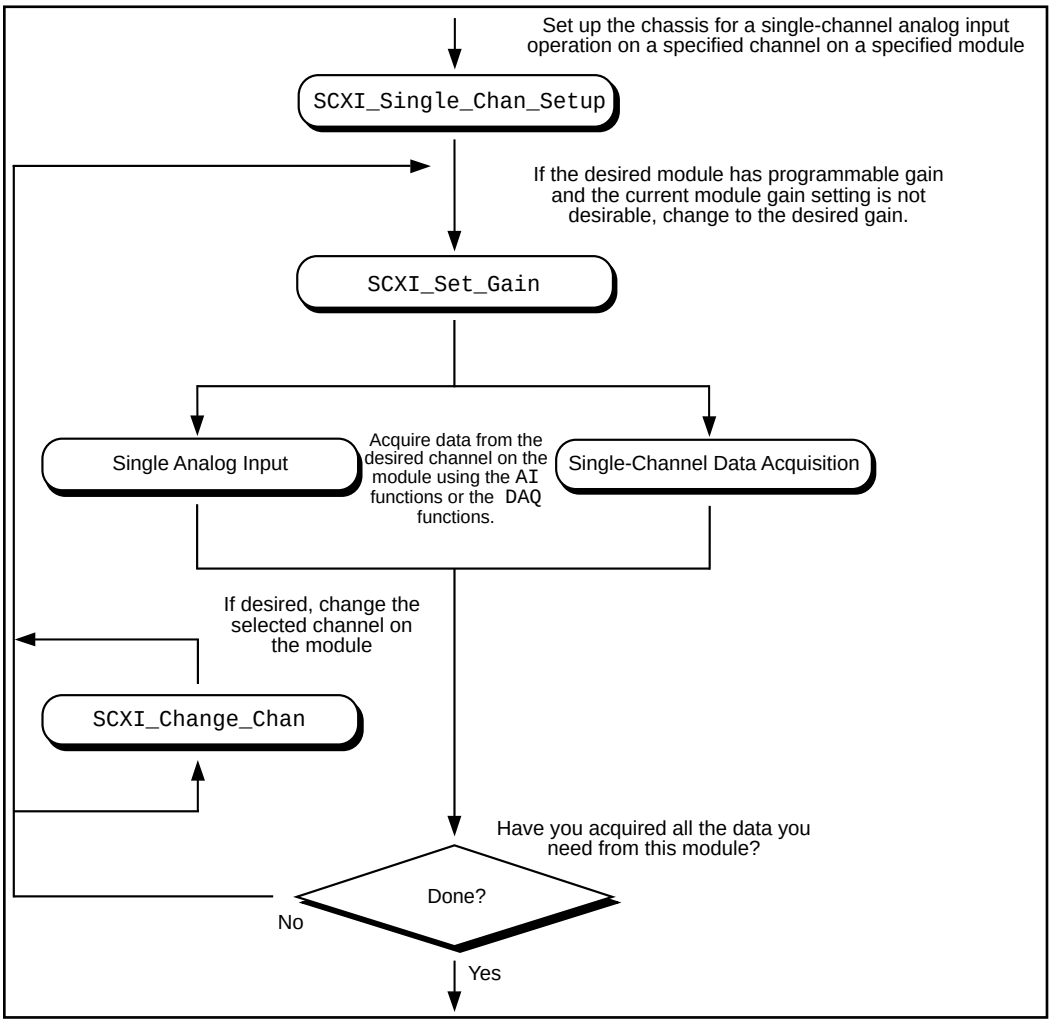


Figure 3-38. Single-Channel or Software-Scanning Operation Using the SCXI-1100, SCXI-1102, SCXI-1120, SCXI-1121, SCXI-1122, or SCXI-1141 in Multiplexed Mode

The `SCXI_Single_Chan_Setup` function selects the given channel to appear at the module output. If the given module is not directly cabled to the DAQ device, the function sends the module output on the SCXIbus and then configures the module that *is* cabled to the DAQ device to send the signal that is present on the SCXIbus to the DAQ device.

The SCXI_Set_Gain function changes the gain of an SCXI-1100, SCXI-1102, SCXI-1122, or SCXI-1141 module. The module maintains this gain setting until you call the function again to change it. You can also do any other module-specific programming at this point, such as SCXI_Configure_Filter or SCXI_Set_Input_Mode.

To achieve software scanning, select a different channel on the module using the SCXI_Change_Chan function after acquiring data from the desired channel with the AI or DAQ functions. If you want a channel on a different module, you must call the SCXI_Single_Chan_Setup function again to enable the appropriate module outputs and manage the SCXIbus signal routing.

Figure 3-39 shows the function call sequence of a single channel or software scanning application using an SCXI-1140 in Multiplexed mode.

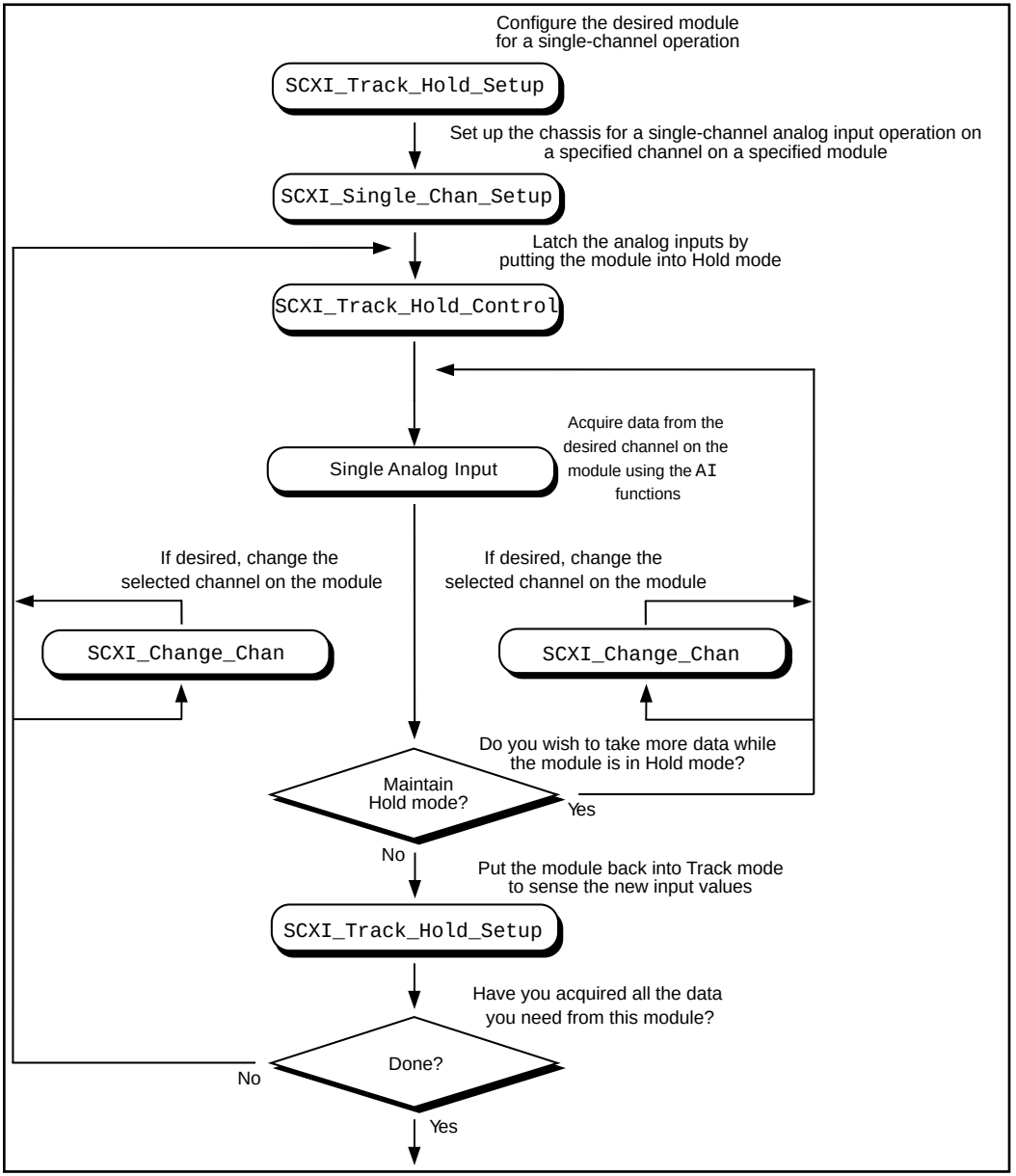


Figure 3-39. Single-Channel or Software-Scanning Operation Using the SCXI-1140 in Multiplexed Mode

Notice the similarities between Figure 3-39 and Figure 3-41, which shows the corresponding application in Parallel mode. The `SCXI_Track_Hold_Setup` calls and the `SCXI_Track_Hold_Control` calls are the same. In Multiplexed mode, however, an `SCXI_Single_Chan_Setup` call is required to select the multiplexed channel and route the output to the DAQ device or SCXI-1200 appropriately. The `SCXI_Change_Chan` call can change the channel on the module either while the module is in Hold mode or after the module has been returned to Track mode.

Figure 3-40 shows the function call sequence of a channel-scanning application in Multiplexed mode. Remember that only the MIO and AI devices, the Lab-PC+, the SCXI-1200, and the DAQCard-1200 support channel scanning in Multiplexed mode. You can use any combination of module types in a scanning operation. If any SCXI-1140 modules are to be scanned, you must use interval scanning; and if you are using a plug-in DAQ device, the module that is directly connected to the DAQ device must be an SCXI-1140.

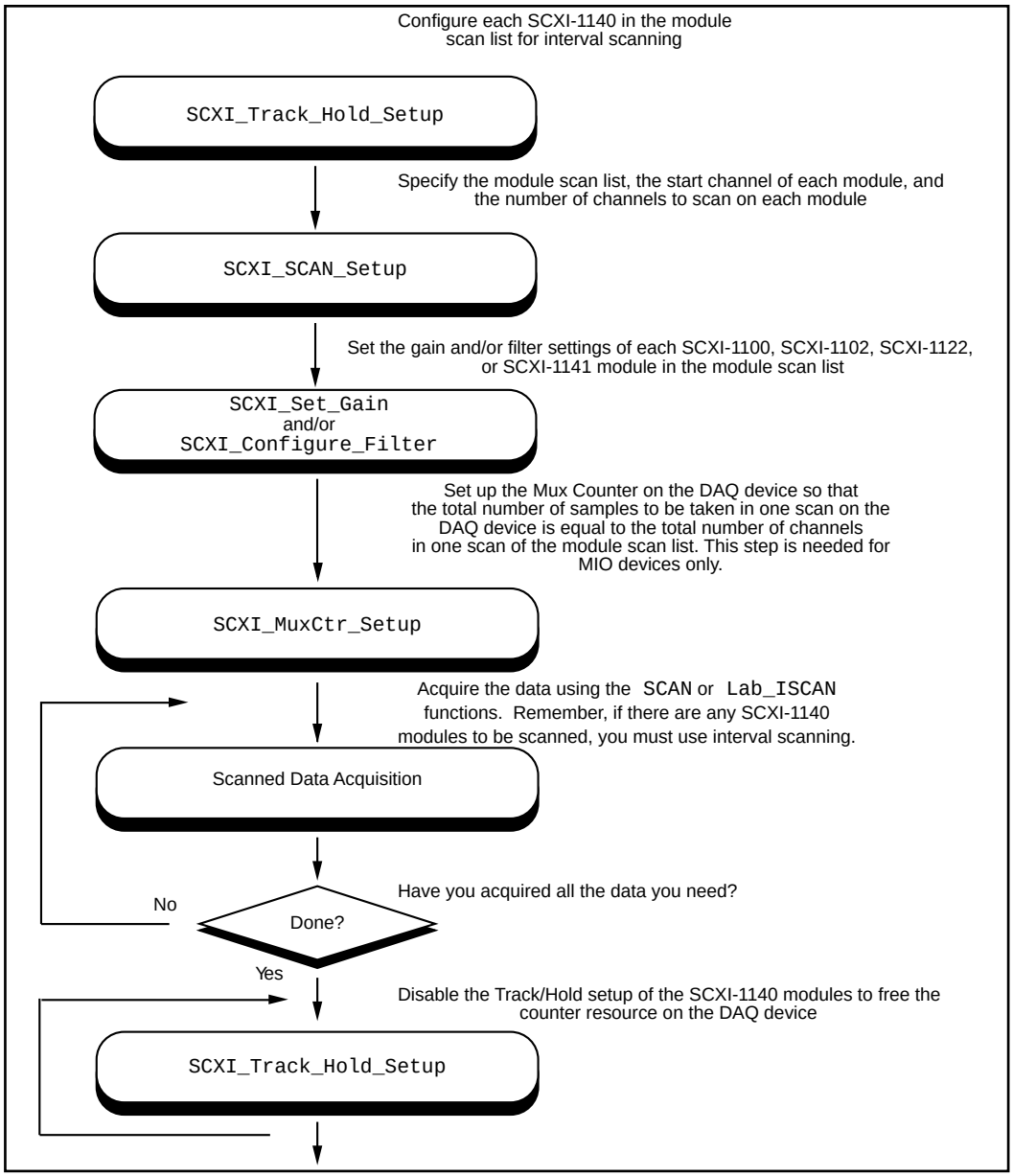


Figure 3-40. Channel-Scanning Operation Using Modules in Multiplexed Mode

If any of the modules to be scanned are SCXI-1140 modules, you must establish the Track/Hold setup of each one. If you want to synchronize multiple SCXI-1140 modules, you can configure the module that is receiving the Track/Hold control signal to send the Track/Hold signal on the SCXIbus so that any other SCXI-1140 modules can use it. The Track/Hold signal can be from either the DAQ device counter or an external source.

The `SCXI_SCAN_Setup` call establishes the module scan list, which NI-DAQ downloads to Slot 0. Each module is programmed for automatic scanning starting at its given start channel. If you will need the SCXIbus during the scan to route the outputs of multiple modules, this function resolves any contention. If you are using an SCXI-1200, you can include the SCXI-1200 in the module scan list.

In many of the data acquisition function descriptions in Chapter 2, *Function Reference*, of the *NI-DAQ Function Reference Manual*, the **count** parameter descriptions specify that **count** must be an integer multiple of the total number of channels scanned. In channel-scanning acquisitions in Multiplexed mode, the total number of channels scanned is the sum of all the elements in the **numChans** array in the `SCXI_SCAN_Setup` function call.

If any of the modules in the module scan list are SCXI-1100, SCXI-1102, SCXI-1122, or SCXI-1141 modules, you can use `SCXI_Set_Gain` to change the gain setting on each module. You can also use the `SCXI_Configure_Filter` function for the SCXI-1122 and SCXI-1141 and the `SCXI_Set_Input_Mode` function for the SCXI-1122.

The `SCXI_MuxCtr_Setup` call synchronizes the module scan list with the DAQ device or SCXI-1200 scan list. In most cases (especially when using interval scanning), it is best to ensure that the number of samples NI-DAQ takes in one pass through the module scan list is the same as the number of samples NI-DAQ takes in one pass through the DAQ device scan list. Please refer to the `SCXI_MuxCtr_Setup` function description in the *NI-DAQ Function Reference Manual*.

After you have set up the SCXI chassis and modules, you can perform more than one channel-scanning operation using the `SCAN` or `Lab_ISCAN` functions without reconfiguring the SCXI chassis or modules.

When you are using the SCXI-1200 to acquire the data, you should pass channel 0 to the Lab_ISCAN functions; the SCXI Slot 0 will take care of all the channel switching.

Building Analog Input Applications in Parallel Mode

When you operate the SCXI-1120, SCXI-1121, and SCXI-1141 modules in Parallel mode, no further SCXI function calls are required beyond those shown in Figure 3-37 to set up the modules for analog input operations. After you have initialized and reset the SCXI chassis and modules, you can use the AI, DAQ, SCAN, or Lab_ISCAN functions with the DAQ device. Remember that the **channel** and **gain** parameters of the AI, DAQ, SCAN, and Lab_ISCAN functions refer to the DAQ device channels and gains.

For example, to acquire a single reading from channel 0 on the module, call the AI_Read function with the **channel** parameter set to 0. The **gain** parameter refers to the DAQ device gain. You can then use the SCXI_Scale function to convert the binary reading to a voltage. The AI_VRead function call is not generally useful in SCXI applications because it does not take into account the gain applied at the SCXI module when scaling the binary reading.

To build a channel-scanning application using the SCXI-1120, SCXI-1121, or SCXI-1141 in Parallel mode, use the SCAN and Lab_ISCAN functions to scan the channels on the DAQ device that correspond to the desired channels on the module. For example, to scan channels 0, 1, and 3 on the module using an MIO-16 device, call the SCAN_Op function with the **channel** vector set to {0, 1, 3}. The **gain** vector should contain the MIO and AI device channel gains. After the data is acquired, you can demultiplex it and send the data for each channel to the DAQ_VScale function. Remember to pass the *total gain* to the DAQ_VScale function to obtain the voltage read at the input of the module.

In many of the data acquisition function descriptions in Chapter 2, *Function Reference*, of the *NI-DAQ Function Reference Manual*, the **count** parameter descriptions specify that **count** must be an integer multiple of the total number of channels scanned. In channel-scanning acquisitions in Parallel mode, the total number of channels scanned is the **numChans** parameter in the SCAN_Setup, SCAN_Op, SCAN_to_Disk, Lab_ISCAN_Start, Lab_ISCAN_Op, or Lab_ISCAN_to_Disk function calls.

When you use the SCXI-1200 module in Parallel mode, you simply use the AI, DAQ, or Lab_ISCAN functions described earlier in this chapter with the logical device number you assigned in the configuration utility. You cannot use the SCXI-1200 to read channels from other analog input modules that are configured for Parallel mode.

The SCXI-1100, SCXI-1102, and SCXI-1122 operate in Multiplexed mode only.

The SCXI-1140 module requires the use of SCXI functions to configure and control the Track/Hold state of the module before you can use the AI,DAQ,SCAN, and Lab_ISCAN functions to acquire the data. Figure 3-41 shows the function call sequence of a single-channel (or software-scanning) operation using the SCXI-1140 module in Parallel mode.

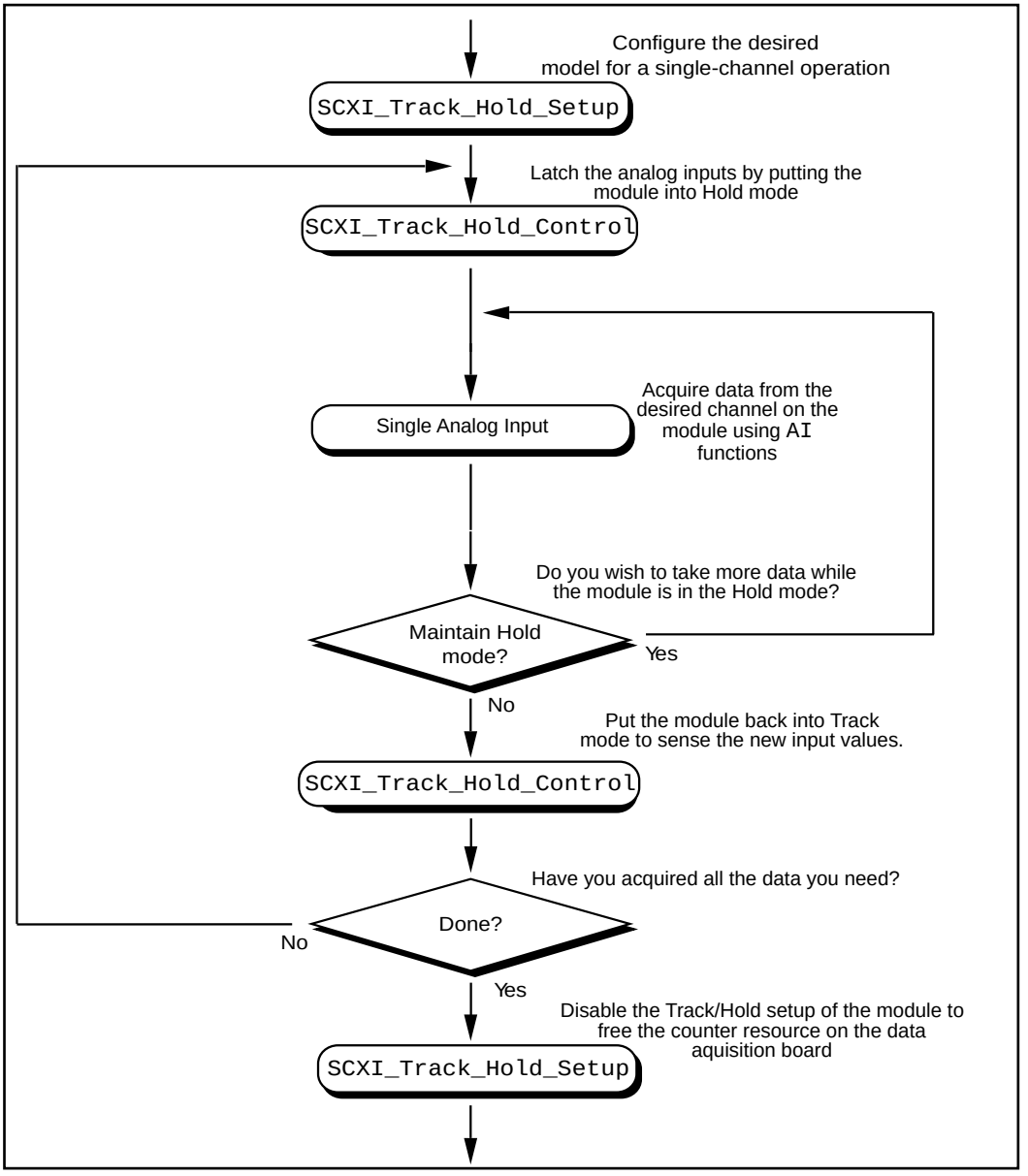


Figure 3-41. Single-Channel or Software-Scanning Operation Using the SCXI-1140 in Parallel Mode

The initial SCXI_Track_Hold_Setup call signals the driver that the module will be used in a single-channel application, and puts the module into Track mode. The first SCXI_Track_Hold_Control call will latch, or sample, all the module inputs; subsequent AI calls will read the voltages that were sampled. It is important to realize that all AI operations that occur between the first SCXI_Track_Hold_Control call, which puts the module into Hold mode, and the second control call, which puts the module into Track mode, acquire data that was sampled at the time of the first control call. One or more channels may be read while the module is in Hold mode. After you put the module back into Track mode, you can repeat the process to acquire new data.

Remember that the **channel** and **gain** parameters of the AI function calls refer to the DAQ device channels and gains. Simply use the data acquisition channels that correspond to the desired module channels as described earlier in this section. You must also be aware of the SCXI-1140 Track/Hold timing requirements that were described in *The SCXI-1140* section of Chapter 2, *Hardware Overview*.

Figure 3-42 shows the function call sequence of a channel-scanning application using the SCXI-1140 in Parallel mode.

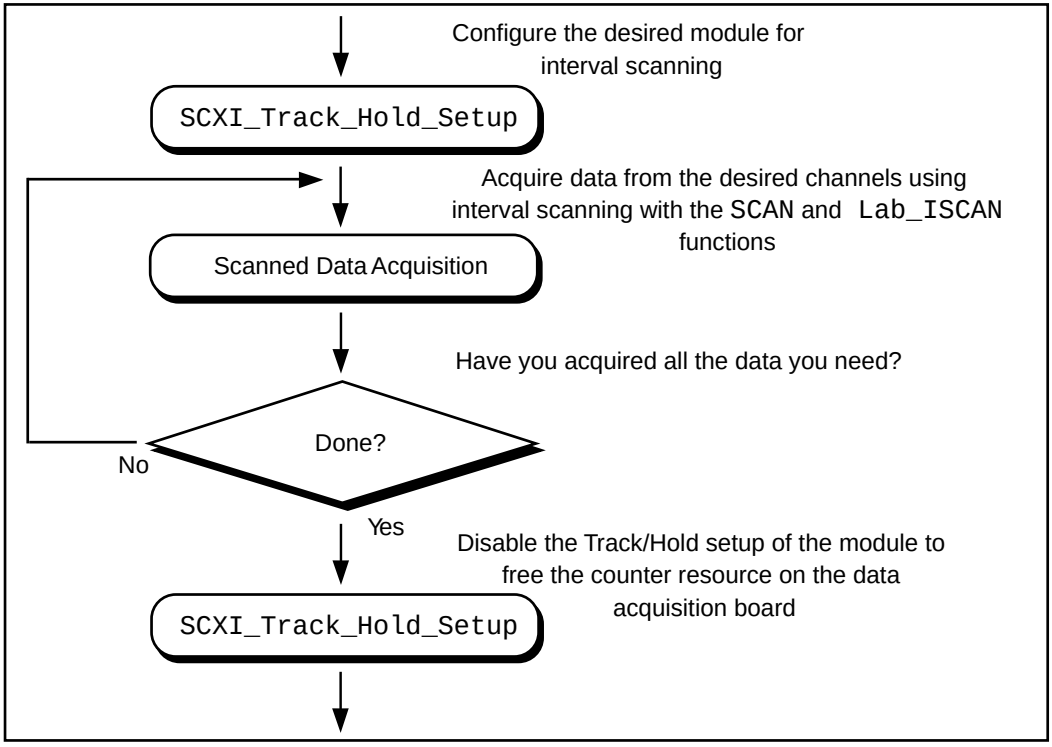


Figure 3-42. Channel-Scanning Operation Using the SCXI-1140 in Parallel Mode

The call sequence is much simpler because the scan interval timer automatically controls the Track/Hold state of the module during the interval-scanning operation. Remember that only the MIO-16, Lab-PC+, SCXI-1200, and DAQCard-1200 devices support channel-scanning using the SCXI-1140 module.

SCXI Data Acquisition Rates

The settling time of the SCXI modules may affect the maximum data acquisition rates that your DAQ device can achieve. The settling times and maximum rates of the different SCXI modules at each gain setting are listed in Table 3-13. If the maximum rate listed for your SCXI

module is *slower* than the applicable maximum rate of your DAQ device, you will have to use the maximum rate listed in Table 3-13.

Table 3-13. Maximum SCXI Data Acquisition Rates

SCXI Module	Gain	Maximum Acquisition Rate	Settling Time
SCXI-1100	1 to 100	143 kS/s	7 µs
	200	100 kS/s	10 µs
	500	62.5 kS/s	16 µs
	1,000, 2,000	20 kS/s	50 µs
SCXI-1102	1	See the *SCXI-1102 User Manual*.	
	100	See the *SCXI-1102 User Manual*.	
SCXI-1120	1 to 2,000	143 kS/s	7 µs
SCXI-1121	1 to 2,000	143 kS/s	7 µs
SCXI-1122	0.01, 0.02, 0.05, 0.5, 0.2, 0.1, 1, 2, 5, 10, 20, 50, 100, 200, 500, 1,000, 2,000	100 kS/s	10 µs
SCXI-1140	1 to 500	143 kS/s	7 µs
SCXI-1200	1, 2, 5, 10, 20	55 kS/s	18 µs
	50	33.3 kS/s	30 µs
	100	33.3 kS/s	50 µs
SCXI-1141	1 to 100	143 kS/s	7 µs

The acquisition rate of the SCXI-1200 module is limited by the rate at which your PC can service interrupts from the parallel port. This is a machine-dependent rate.

The filter setting on the SCXI-1100 and the SCXI-1122 will dramatically affect settling time. See Appendix A in your *SCXI-1100 User Manual* or *SCXI-1122 User Manual* for details.

☞ ***Note:*** ***The SCXI-1122 uses relays to switch the input channels; the relays require 10 ms to switch, so the sampling rate in a channel scanning operation cannot exceed 100 Hz. If you want to take many readings from each channel and average them to reduce noise, you should use the single-channel or software-scanning method shown in figure 3-40 instead of the channel-scanning method shown in figure 3-42. This means you select one channel on the module, acquire many samples on that channel using the*** DAQ ***functions, select the next channel, and so on. This will increase the lifetime of your module relays. Once you have selected a particular channel, you can use the fastest sample rate your DAQ device supports with the*** DAQ ***functions.***

Analog Output Applications

Using the SCXI-1124 analog output module with the NI-DAQ functions is very simple. Just call the `SCXI_AO_Write` function to write your desired voltages to the DAC channels on the module. You can use the `SCXI_Get_Status` function, if you wish, to determine when the DAC channels have settled to their final analog output voltages.

If you want to calculate new calibration constants for `SCXI_AO_Write` to use for the voltage to binary conversion instead of the factory calibration constants that are shipped in the module EEPROM, follow the procedure outlined in the `SCXI_Cal_Constants` function description.

Digital Applications

If you configured your digital or relay modules for Multiplexed mode, use the `SCXI_Set_State` and `SCXI_Get_State` functions to access your digital or relay channels.

If you are using the SCXI-1160 module, you may wish to use the `SCXI_Get_Status` function after calling the `SCXI_Set_State` function. `SCXI_Get_Status` will tell you when the SCXI-1160 relays have finished switching.

If you are using the SCXI-1162 or SCXI-1162HV module, `SCXI_Get_State` will read the module input channels. For the other digital and relay modules, `SCXI_Get_State` will return a software copy of the current state that NI-DAQ maintains. However, if you are using the SCXI-1163 or SCXI-1163R in Parallel mode, `SCXI_Get_State` will read the hardware states.

If you are using the SCXI-1162, SCXI-1162HV, SCXI-1163, or SCXI-1163R in Parallel mode, you can use the SCXI functions as described above, or you can call the `DIG_In_Port` and `DIG_Out_Port` functions using the correct DAQ device port numbers that correspond to the SCXI module channels. *The DIO-24 and DIO-96* and *The DIO-32F* sections in Chapter 2, *Hardware Overview*, list the onboard port numbers that are used for each type of device if the SCXI-1162, SCXI-1162HV, SCXI-1163, or SCXI-1163R is configured for Parallel mode. The MIO and AI devices, Lab-PC+, and SCXI-1200 cannot use the SCXI-1162, SCXI-1162HV, SCXI-1163, or SCXI-1163R in Parallel mode.

The Transducer Conversion Functions

Source code for transducer conversion functions is included with NI-DAQ. The Transducer Conversion functions convert analog input voltages read from thermocouples, RTDs, thermistors, and strain gauges into units of temperature or strain:

`RTD_Convert`	Both single-voltage and voltage-buffer routines are supplied that convert voltages read from an RTD into resistance and then into temperature in Celsius, Fahrenheit, kelvin, or Rankine.
`Strain_Convert`	Both single-voltage and voltage-buffer routines are supplied that convert voltages read from a strain gauge into measured strain using the formula appropriate to the strain gauge bridge configuration used.
`Thermistor_Convert`	Both single-voltage and voltage-buffer routines are supplied that convert voltages read from thermistors into temperature.
`Thermocouple_Convert`	Both single-voltage and voltage-buffer routines are supplied that convert voltages read from E-, J-, K-, R-, S-, or T-type thermocouples into temperature

in Celsius, Fahrenheit, kelvin, or Rankine.

NI-DAQ for PC compatibles installs the source files for these functions in the same directories as the example programs. You can cut and paste, include, or merge these conversion routines into your application source files so that you can call the routines in your application. One of the SCXI example programs includes the thermocouple conversion routine, so you can refer to that program to see how the conversion is incorporated into an application.

The conversion routines were included in NI-DAQ as source files rather than driver function calls so that you have complete access to the conversion formulas. You can edit the conversion formulas or replace them with your own to meet the specific accuracy requirements of your application. Comments in the conversion source code facilitate any editing you feel is necessary.

There is a header file for each language that contains the constant definitions used in the conversion routines. Include or merge this header file into your application program.

The transducer conversion routine descriptions apply to all languages.

Transducer Conversion Function Descriptions

RTD_Convert

RTD_Buf_Convert

Purpose

Converts a voltage or voltage buffer that NI-DAQ read from an RTD into temperature.

Parameter Discussion

convType is an integer that indicates whether to use the given conversion formula, or to use a user-defined formula that you have put into the routine.

0: Use the given conversion formula.
-1: Use a user-defined formula that has been added to the routine.

Iex is the excitation current that was used with the RTD. If a 0 is passed in **Iex,** a default excitation current of 0.15 mA is assumed.

Ro is the RTD resistance at 0° C.

A and **B** are the coefficients of the Callendar Van-Dusen equation that fit your RTD.

TempScale is an integer indicating in which temperature units you want your return values to be. Constant definitions for each temperature scale are given in the conversion header file.

1: Celsius
2: Fahrenheit
3: Kelvin
4: Rankine

The `RTD_Convert` routine has two remaining parameters—**RTDVolts** is the voltage that NI-DAQ read from the RTD, and **RTDTemp** is the return temperature value.

The `RTD_Buf_Convert` routine has three remaining parameters—**numPts** is the number of voltage points to convert, **RTDVoltBuf** is the array that contains the voltages that NI-DAQ read from the RTD, and **RTDTempBuf** is the return array that will contain the temperatures.

Using This Function

The conversion routines first find the RTD resistance by dividing **RTDVolts** (or each element of **RTDVoltBuf**) by **Iex**. The function converts that resistance to a temperature using a solution to the Callendar Van-Dusen equation for RTDs:

$$Rt = Ro[1 + At + Bt^2 + C(t-100)t^3]$$

For temperatures above 0° C, the C coefficient is 0 and the equation reduces to a quadratic equation for which we have found the appropriate root. Thus, these conversion routines are accurate only for temperatures above 0° C.

Your RTD documentation should give you **Ro** and the **A** and **B** coefficients for the Callendar Van-Dusen equation. The most common RTDs are 100 Ω platinum RTDs that either follow the European temperature curve (also known as the DIN 43760 standard) or the American curve. The values for **A** and **B** are as follows:

- European Curve (DIN 43760):

  A = 3.90802×10^{-3}

  B = -5.80195×10^{-7}

  ($\alpha = 3.85 \times 10^{-3}$; $\partial = 1.492$)

- American Curve:

  A = 3.9784 x 10^{-3}

  B = -5.8408 x 10^{-7}

  (α = 3.92 x 10^{-3}; ∂ = 1.492)

Some RTD documentation contains values for α and ∂, from which you can calculate **A** and **B** using the following equations:

$$\mathbf{A} = \alpha(1 + \partial/100)$$

$$\mathbf{B} = -\alpha\,\partial/100^2$$

where α is the temperature coefficient at T = 0° C.

Strain_Convert

Strain_Buf_Convert

Purpose

Converts a voltage or voltage buffer that NI-DAQ read from a strain gauge to units of strain.

Parameter Discussion

bridgeConfig is an integer indicating in what type of bridge configuration the strain gauge is mounted. Figure 3-43 shows all the different bridge configurations and the corresponding values that you should pass in **bridgeConfig**.

Vex is the excitation voltage that you used. If the value of **Vex** is 0, a default excitation voltage of 3.333 V is assumed. The SCXI-1121 module provides for excitation voltages of 10 V and 3.333 V. The SCXI-1122 module provides for an excitation voltage of 3.333 V.

GF is the gauge factor of the strain gauge.

v is Poisson's Ratio (only needed in certain bridge configurations).

Rg is the strain gauge nominal value.

RL is the lead resistance. In many cases, the lead resistance is negligible and you can pass a value of 0 for **RL** to the routine. Otherwise, you can measure **RL** to be more accurate.

Vinit is the unstrained voltage of the strain gauge after it has been mounted in its bridge configuration. You should read this voltage at the beginning of your application and save it to pass to the strain gauge conversion routines.

The `Strain_Convert` routine has two remaining parameters—**strainVolts** is the voltage that NI-DAQ read from the strain gauge, and **strainVal** is the return strain value.

The `Strain_Buf_Convert` routine has three remaining parameters—**numPts** is the number of voltage points to convert, **strainVoltBuf** is the array that contains the voltages that NI-DAQ read from the strain gauge, and **strainValBuf** is the return array that will contain the strain values.

Using This Function

The conversion formula used is based solely on the bridge configuration. Figure 3-43 shows the seven bridge configurations supported and the corresponding formulas. For all bridge configurations, NI-DAQ uses the following formula to obtain Vr:

Vr = (**strainVolts** - **Vinit**) / **Vex**

In the circuit diagrams shown in Figure 3-43, VOUT is the voltage you measure and pass to the `Strain_Convert` function as the **strainVolts** parameter. In the quarter-bridge and half-bridge configurations, R_1 and R_2 are dummy resistors that are not directly incorporated into the conversion formula. The SCXI-1121 and SCXI-1122 modules provide R_1 and R_2 for a bridge-completion network, if needed. Refer to your *Getting Started with SCXI* manual for more information on bridge-completion networks and voltage excitation.

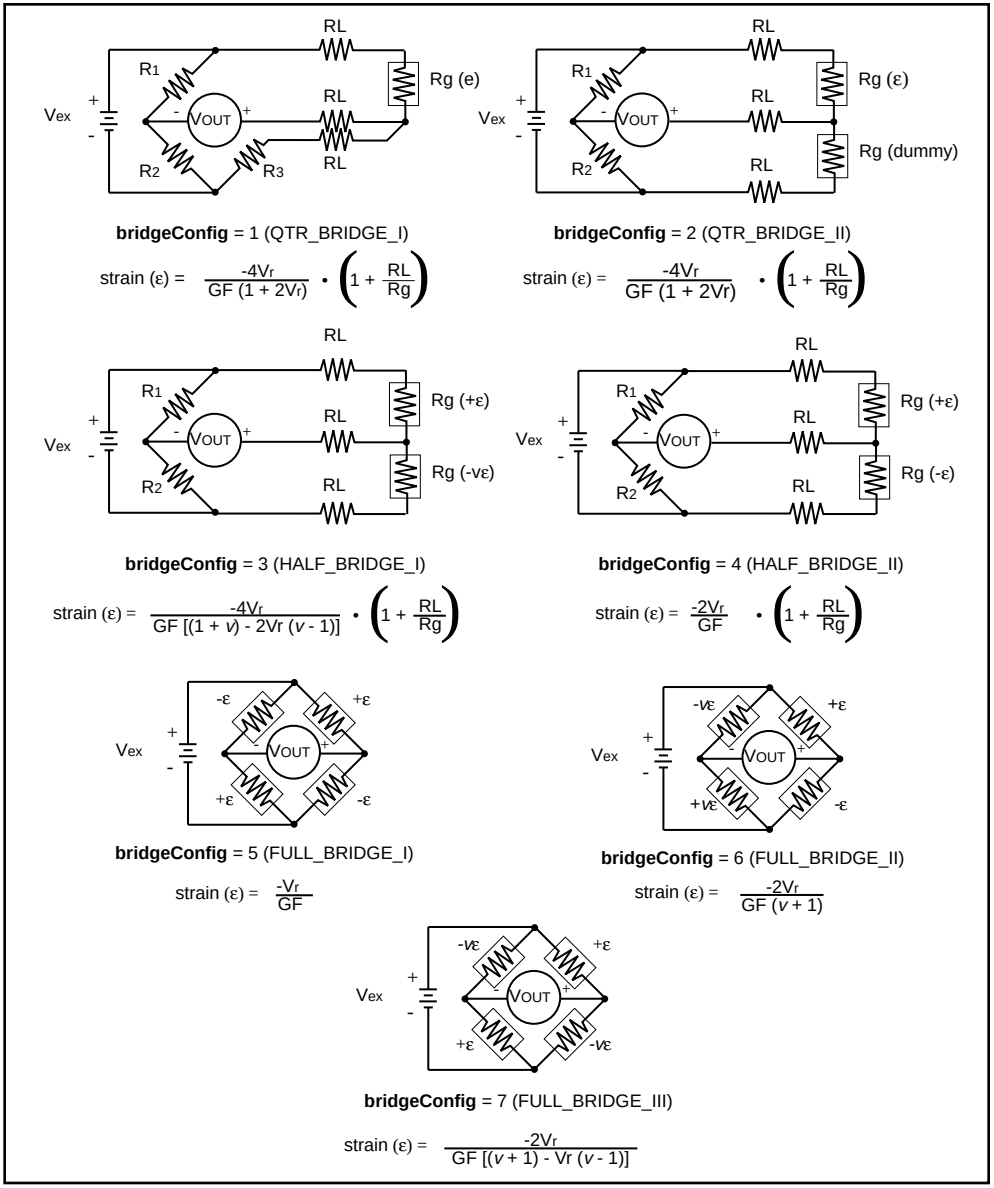


Figure 3-43. Strain Gauge Bridge Configuration

Thermistor_Convert

Thermistor_Buf_Convert

Purpose

Converts a voltage or voltage buffer that was read from a thermistor to temperature. Some SCXI terminal blocks have onboard thermistors that you can use to do cold-junction compensation.

Parameter Discussion

Vref is the voltage reference you apply across the thermistor circuit (see Figure 3-44). The thermistor on the SCXI terminal blocks has a **Vref** of 2.5 V.

R1 is the value expressed in Ohms of the resistor in series with your thermistor (see Figure 3-44). The thermistor on the SCXI terminal blocks has an **R1** value of 5,000 Ω.

TempScale is an integer indicating in which temperature unit you want your return values to be. Constant definitions for each temperature scale are given in the conversion header file.

1: Celsius
2; Fahrenheit
3: Kelvin
4: Rankine

The `Thermistor_Convert` function has two remaining parameters—**Volts** is the voltage that you read from the thermistor, and **Temperature** is the return temperature value given in units determined by **TempScale**.

The `Thermistor_Buf_Convert` function has three remaining parameters—**numPts** is the number of voltage points to convert, **VoltBuf** is the array of voltages that you read from the thermistor, and **TempBuf** is the return array of temperature values given in units determined by **TempScale**.

Using This Function

The following equation expresses the relationship between **Volts** and `Rt`, the thermistor resistance (see Figure 3-44).

Volts = **Vref** (`Rt` / (**R1** + `Rt`))

Solving the previous equation for `Rt`, we have:

`Rt` = **R1** (**Volts** / (**Vref** - **Volts**))

Once we calculate Rt, we use the following equation to convert Rt, the thermistor resistance, to temperature in Kelvin. We then convert the temperature to the desired temperature scale if necessary.

T = 1 / (a + b(ln Rt) + c(ln Rt)^3)

The values used for a, b, and c are given below. These values are correct for the thermistors provided on the SCXI terminal blocks. If you are using a thermistor with different values for a, b, and c (consult you thermistor data sheet), you can edit the thermistor conversion routine to use your own a, b, and c values.

a = 1.295361E-3

b = 2.343159E-4

c = 1.018703E-7

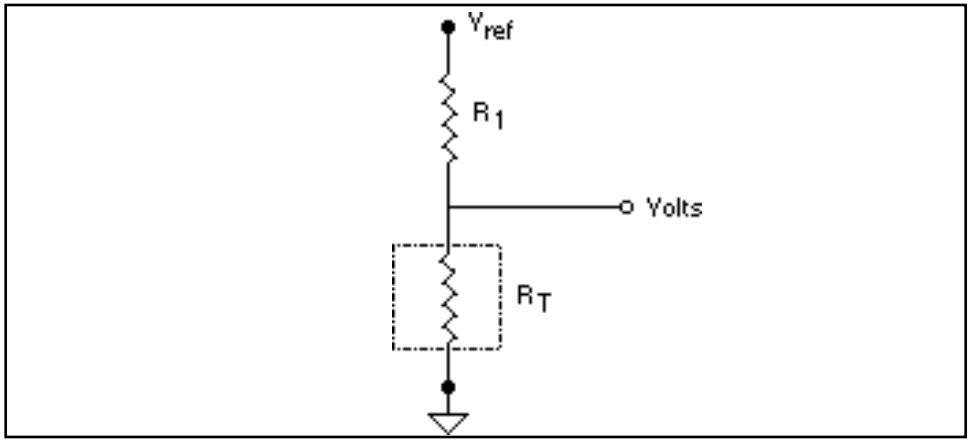


Figure 3-44. Circuit Diagram of a Thermistor in a Voltage Divider

Thermocouple_Convert

Thermocouple_Buf_Convert

Purpose

Converts a voltage or voltage buffer that NI-DAQ read from a thermocouple into temperature.

Parameter Discussion

TCType is an integer indicating what type of thermocouple NI-DAQ used to read the temperature. Constant definitions for each thermocouple type are given in the conversion header file. You can use the constants that have been defined, or you can pass integer values to the routine.

1: E
2: J
3: K
4: R
5: S
6: T
7: B
8: N

CJCTemp is the temperature in Celsius that NI-DAQ will use for cold-junction compensation of the thermocouple temperature. If you are using SCXI, this will most likely be the temperature that NI-DAQ read from the temperature sensor on the SCXI terminal block. The AMUX-64T also has a temperature sensor that you can use for this purpose.

TempScale is an integer indicating in which temperature unit you want your return values to be. Constant definitions for each temperature scale are given in the conversion header file.

1: Celsius
2: Fahrenheit
3: Kelvin
4: Rankine

The `Thermocouple_Convert` routine has two remaining parameters—**TCVolts** is the voltage that NI-DAQ read from the thermocouple, and **TCTemp** is the return temperature value.

The `Thermocouple_Buf_Convert` routine has three remaining parameters—**numPts** is the number of voltage points to convert, **TCVoltBuf** is the array that contains

the voltages that NI-DAQ read from the thermocouple, and **TCTempBuf** is the return array that will contain the temperatures.

Using This Function

These routines convert **TCVolts** (or each element of **TCVoltBuf**) into a corresponding temperature after performing the necessary cold-junction compensation. Cold-junction compensation is done by converting **CJCTemp** into an equivalent thermocouple voltage and adding it to **TCVolts**. The actual temperature-to-voltage conversion is done by choosing the appropriate reference equation that characterizes the correct temperature subrange for the specific **TCType**. The valid temperature range for a given **TCType** is divided into several subranges with each subrange characterized by a reference equation. The computed voltage is then added to **TCVolts** to perform the cold-junction correction. The conversion of **TCVolts** into a corresponding temperature is done by using inverse equations that are specified for a given **TCType** for different subranges. These inverse equations have an error tolerance as shown in Table 3-14. All the reference equations and inverse equations used in these routines are from *NIST Monograph 175*.

Table 3-14 shows the valid temperature ranges and accuracies for the inverse equations used for each thermocouple type. The errors listed in the table refer to the equations only; they do not take into consideration the accuracy of the thermocouple itself, the SCXI modules, or the DAQ device that is used to take the voltage reading.

Table 3-14. Temperature Error for Thermocouple Inverse Equations

Thermocouple Type	Temperature Range	Error
B	250° to 700° C 700° to 1,820° C	-0.02° to +0.03° C -0.01° to +0.02° C
E	-200° to 0° C 0° to 1,000° C	-0.01° to +0.03° C ±0.02° C
J	-210° to 0° C 0° to 760° C 760° to 1,200° C	-0.05° to +0.03° C ±0.04° C -0.04° to +0.03° C
K	-200° to 0° C 0° to 500° C 500° to 1,372° C	-0.02° to +0.04° C -0.05° to +0.04° C -0.05° to +0.06° C
N	-200° to 0° C 0° to 600° C 600° to 1,300° C	-0.02° to +0.03° C -0.02° to +0.03° C -0.04° to +0.02° C

Table 3-14. Temperature Error for Thermocouple Inverse Equations (Continued)

Thermocouple Type	Temperature Range	Error
R	-50° to 250° C 250° to 1,200° C 1,200° to 1,664.5° C 1,664.5° to 1,768.1° C	±0.02° C ±0.005° C -0.0005° to +0.001° C -0.001° to +0.002° C
S	-50° to 250° C 250° to 1,200° C 1,200° to 1,664.5° C 1,664.5° to 1,768.1° C	±0.02° C ±0.01° C ±0.0002° C ±0.002° C
T	-200° to 0° C 0° to 400° C	-0.02° to +0.04° C ±0.03° C

DMA and Programmed I/O Performance Limitations

Chapter 4

This chapter discusses data acquisition performance reductions caused by interrupt latency in the Windows programming environment.

`DIG_Block`, `DAQ`, `SCAN`, `MDAQ`, and `WFM` operations all input or output blocks of data to or from a plug-in DAQ device. For input operations, NI-DAQ must transfer the incoming data to a buffer in the computer memory. For output operations, NI-DAQ must transfer outgoing data from a buffer in the computer memory to the DAQ device. NI-DAQ uses two mechanisms to perform the data transfer. The first option, programmed I/O, transfers each data point through software. The second option is to use the DMA controller chip to perform a hardware transfer of the data. The speed of analog and digital input and output operations is limited by the transfer mechanism as well as by the computer, board, and operating system. This chapter explains the performance limitations for DOS and Windows applications.

An Explanation of Programmed I/O and DMA Transfers

Programmed I/O is a software-intensive method for transferring data from computer memory to an I/O device, in this case a data acquisition plug-in board. For each data point, the CPU must execute code that transfers data to the board. Therefore, the CPU is tied up while data is being written to or read from the board. The CPU is free to execute other code, including applications, when it is not writing or reading data to or from the board.

NI-DAQ utilizes interrupt service routines to do background transfers to DAQ devices. The CPU is interrupted to do data transfers only when the board asserts an interrupt indicating it is ready for the next data point to be read or written.

In contrast, DMA transfers use hardware rather than software to transfer data between computer memory and the board. This is accomplished by programming the DMA controller chip. The DMA chip performs the transfers between memory and I/O devices

This document was created with FrameMaker 4.0.4

independently of the CPU. As a result, the CPU is freed from having to execute code to transfer each individual data point, making it available for execution of your applications. Of course, the CPU and DMA share control of the same bus, so some decline in computer performance may occur even when using DMA transfers.

Programmed I/O or DMA

Whether NI-DAQ uses programmed I/O or DMA depends on the board and the transfer mode that you select. If you have an analog input and/or output board or the AT-DIO-32F, refer to the `Set_DAQ_Device_Info` function description in Chapter 2, *Function Reference*, of the *NI-DAQ Function Reference Manual* to find out the transfer mode you will be using. Boards that use interrupts use programmed I/O.

The following boards use programmed I/O for block digital input and output:

- AT-MIO-16D and AT-MIO-16DE-10
- Lab-PC+, SCXI-1200, DAQPad-1200, and DAQCard-1200
- DIO-24
- PC-DIO-96

The following board uses DMA for block digital input and output:

- AT-DIO-32F

Using DMA on AT Bus Computers

Page Boundaries in AT Bus Computers

On AT bus computers, the DMA controller organizes computer memory addresses into pages. When performing 16-bit data transfers, the DMA can access up to 128 KB of system memory between page boundaries. If a data buffer spans a DMA page boundary, you must reprogram the DMA controller to continue DMA transfers on the next memory page. On many of the data acquisition adapters, hardware FIFOs on the adapter serve as buffers, and provide adequate time to reprogram the DMA controller without disrupting the acquisition. However, the lack of a hardware FIFO on the AT-MIO-16F-5 analog output prohibits the board from operating at the maximum rate if the

data buffer contains one or more page boundaries. In this case, the time between each DMA transfer (for example, the update interval in waveform generation) must be more than the time needed for the reprogramming. A similar situation arises if you are using an AT-DIO-32F, and you have selected the transfer mode so that group 1 and group 2 each has one DMA channel available for digital input/output.

When you call `WFM_Load` to perform waveform generation on an AT-MIO-16F-5 using DMA, the function checks if the waveform buffer contains any DMA page breaks. If so, a **pageBreakInBuffer** warning is returned.

When configured for a pattern generation, `DIG_Block_In` and `DIG_Block_Out` return the same warning if any DMA page breaks exist in the pattern-generation buffer.

Depending on the requirement of your application, choose one of the following approaches to deal with the possibility of page boundaries when using an AT-MIO-16F-5 or an AT-DIO-32F:

1. Always treat a buffer as if it contains DMA page breaks, and limit the minimum update interval to be appropriate for buffers with page breaks. This approach ensures that you can use any buffers successfully.
2. Prepare the buffers so that the data does not cross any DMA boundaries. The `Align_DMA_Buffer` function can shift the data to a region within the buffer that does not contain DMA page breaks. `Align_DMA_Buffer` tries to perform the alignment as long as the buffer size is greater than the amount of data in the buffer. However, to guarantee a successful alignment, the buffer size should be at least twice as large as the data set. Realigning a buffer allows maximum speed performance but limits the size of the waveform or pattern to half of the largest allocatable buffer size. There is no restriction on using both aligned and unaligned buffers in the same application.
3. If you are using an AT-DIO-32F and want to use group 1 for digital input/output, you can set the transfer mode so that both DMA channels are available to group 1.

Using Physical Memory Above 16 MB on ISA Bus Computers

NI-DAQ can use DMA to transfer data to and from buffers above 16 MB of physical memory on an ISA bus computer under Windows

and Windows NT environments. Typically, this is a limitation because the DMA controller on ISA bus computers cannot transfer data to physical memory above the 16 MB address range. NI-DAQ will use an intermediate (mirror) buffer to transfer data to and from the DAQ device and then copy data to the user buffer above 16 MB. To take advantage of this feature in your DAQ application under Windows, you must have `NIVISRD.386` installed on your computer.

You can have single or multiple devices doing multiple DMA transfers at the same time to or from memory above 16 MB. You do not need to change your application to take advantage of this feature.

NI-DAQ allocates 4 KB long physically contiguous mirror buffers below 16 MB of physical memory for each DMA channel in the system. Under Windows, this happens at Windows startup time, and under Windows NT, when Windows NT boots. This is done to increase the possibility of successfully allocating a mirror buffer.

When doing DMA at run time, if NI-DAQ finds any part of the user buffer crossing the 16 MB DMA boundary, it does DMA into the mirror buffer associated with the DMA channel already in use. If no mirror buffer can be allocated for that particular DMA channel at startup time, NI-DAQ returns an **invalidMemRegionErr** (-199) error to your application. NI-DAQ then copies data from the mirror buffer into the user buffer when using DMA to transfer data into the system memory from your device, and from user buffer to mirror buffer when using DMA to transfer data from the system memory to your device. Because of this copying, there may be a drop in performance when NI-DAQ is trying to transfer data using DMA into memory above 16 MB on ISA bus computers.

We encourage you to use two DMA channels whenever possible to take advantage of this feature and still achieve maximum performance. The mirror buffers are not used when the user buffer lies below 16 MB of physical memory.

General Performance Considerations for DOS

The DOS operating system imposes almost no overhead on applications. Therefore, the performance of your application is primarily limited by the speed of the computer and the board used. High-speed computers generally allow for faster input and output

operations. Of course, the DAQ device itself will set the absolute maximum limits of performance.

General Performance Considerations for Windows

Interrupt latency in Windows can impose performance limitations on data acquisition. The magnitude of the performance reduction depends on the board, the method used to acquire the data (programmed I/O versus DMA), and most importantly, the mode in which Windows is operating (real, standard, or enhanced).

Interrupt latency is the delay between the time hardware asserts an interrupt and when the interrupt service routine is activated. In DOS, the interrupt latency is minimal because the hardware transfers control directly to the interrupt service routine. In Windows, however, system software transfers control to the interrupt service routine, imposing a software delay.

The size of the software delay depends on the Windows mode. Interrupt latency in Windows Real mode is only slightly more than in DOS. In Standard mode, the interrupt latency becomes more significant, and there is a noticeable reduction in performance from Real mode. The interrupt latency in Windows 386 Enhanced mode is dramatically higher than in the other Windows modes. In fact, the transfer software delay makes up the majority of the time spent in servicing an interrupt, not the interrupt service routine itself.

Programmed I/O Performance in Windows

Interrupt latency hampers programmed I/O acquisition because of the additional delay it imposes before data can be input or output to or from a board by the interrupt service routine. The length of the delay is directly dependent on the speed of the computer.

Generally, individual computer performance governs the maximum programmed I/O rate that you can achieve. As your computer approaches the limit of programmed I/O performance, the CPU spends the majority of time servicing interrupts. As a result, the Windows user interface performance becomes sluggish. The performance limit for Windows 386 Enhanced mode is significantly lower than the performance achieved in Real or Standard mode.

DMA Performance in Windows

Buffers Requiring Reprogramming

Interrupt latency can slow data acquisition that uses DMA when DMA reprogramming is required. NI-DAQ may have to reprogram the DMA controller for four reasons:

- The DMA controllers for AT bus computers organize memory addresses into 64 K word pages. If a data buffer spans one of the page boundaries, the DMA controller must be reprogrammed to continue the DMA transfer on the next memory page.
- In addition, large buffers on any platform may require reprogramming caused by limitations on the transfer counts that can be written to the DMA controllers. On AT bus computers, the DMA transfer count is limited to 16 bits, or 64 KB. EISA bus computers allow counts of 24 bits or 16 MB, which should be sufficient to avoid the problem in most cases.
- Reprogramming may be required because of the virtual memory management system used in Windows 386 Enhanced mode. When a buffer is locked into physical memory in preparation for a data acquisition operation, the buffer may be fragmented if the memory manager cannot find a large enough contiguous space in memory. Each separate piece of the buffer requires DMA reprogramming.

Why Reprogramming Limits Performance

Reprogramming the DMA controller limits performance because it can cause significant pauses between data transfer requests from the DMA controller. Pauses during high-speed input operations can cause acquisition boards to miss or overwrite data points. For output operations, pauses may result in glitches in waveform or pattern generations (time lapses greater than the programmed period between data points). The maximum length of the pause is equal to the interrupt latency plus the time to reprogram the DMA controller. Therefore, data acquisition has the longest pauses in Windows 386 Enhanced mode and the shortest pauses in Real mode because of the differences in interrupt latency.

Results of Performance Limitation

DMA performance limitations manifest themselves in different ways.

DMA reprogramming for analog input may cause a FIFO overflow error (**-75 overFlowErr**). FIFO overflows occur when an analog input board is forced to overwrite unretrieved data in the FIFO. This happens during DMA reprogramming when there is a pause between DMA transfer requests to the board.

DMA reprogramming for analog waveform generation may cause update errors (**-153 dacUpdateErr**). Update errors occur when the analog output board cannot update the output voltage because the DMA controller has not transferred the next data point to the board. Again, the reason for this delay is the pause in DMA transfer requests caused by reprogramming.

With digital input and output boards that use DMA, transfers using handshaking are not affected by the DMA reprogramming, aside from possibly causing the transfer to be slower. On the other hand, DMA reprogramming affects digital pattern generation. Because digital I/O boards do not detect overflow or update errors; however, NI-DAQ cannot explicitly warn you when DMA reprogramming is causing a performance problem during pattern generation. For this reason, the `DIG_Block_In` and `DIG_Block_Out` functions return a warning when DMA reprogramming is required (**15 DMAReprogramming**).

Methods for Eliminating Performance Limitations

You can eliminate performance limitations due to DMA reprogramming several ways:

- A FIFO on a DAQ device can buffer data during reprogramming. However, high-speed acquisitions can easily overflow a small onboard buffer during reprogramming if the interrupt latency is significant (that is, when operating in Standard or Enhanced mode).
- Another way to avoid DMA performance limitations is to use a specialized DMA reprogramming method such as chaining or channel switching. These methods effectively eliminate the reprogramming problem most of the time. For EISA bus computers, NI-DAQ uses chaining in all but two cases, and therefore experiences no performance limitations (exceptions–double-buffered digital I/O, and waveform generation with partial transfers enabled). NI-DAQ also uses channel switching on AT bus computers for the MIO and E Series devices.
- Service interrupts through `NIVISRD.386` in Windows 386 enhanced mode. `NIVISRD.386` minimizes interrupt latency by

intercepting interrupts at the Windows kernel level. Therefore, DMA can be reprogrammed as soon as possible to minimize the possibility of FIFO overflow errors.

- If you are using an AT-DIO-32F and want to use group 1 for digital input/output, you can set the transfer mode so that both DMA channels are available to group 1.

Methods You Can Use to Avoid DMA Reprogramming

You can eliminate the need for DMA reprogramming as follows:

- On an AT bus computer, you can eliminate DMA page breaks. The `Align_DMA_Buffer` function attempts to eliminate breaks in the buffer of this type. This function is explained in the *Page Boundaries in AT Bus Computers* section earlier in this chapter.
- Use buffers small enough to avoid the transfer count limitations of AT and EISA bus computers.
- Avoid using large buffers when operating in Windows 386 Enhanced mode. Large buffers increase the odds of getting a discontiguous buffer that requires DMA reprogramming.

NI-DAQ Double Buffering

Chapter 5

This chapter describes using double-buffered data acquisition with NI-DAQ.

Overview

Conventional data acquisition software techniques, such as single-buffered data acquisition, work well for most of today's applications. However, more sophisticated applications involving larger amounts of data input or output at higher rates require more advanced techniques for managing the data. One such technique is double buffering. National Instruments uses double-buffering techniques in its driver software for continuous, uninterrupted input or output of large amounts of data.

This chapter discusses the fundamentals of double buffering, including specific information on how the NI-DAQ double-buffered functions work.

☞ **Note:** ***Input and output refer to both digital and analog operations in this chapter.***

Single-Buffered Versus Double-Buffered Data

The most common method of data buffering found in conventional driver software is single buffering. In single-buffered input operations, a fixed number of samples are acquired at a specified rate and transferred into computer memory. After the data is stored into the memory buffer, the computer can analyze, display, or store the data to the hard disk for later processing. Single-buffered output operations output a fixed number of samples from computer memory at a specified rate. After the data is output, the buffer can be updated with new or freed data.

Single-buffered operations are relatively simple to implement, can usually take advantage of the full hardware speed of the DAQ device, and are very useful for many applications. The major disadvantage of

single-buffered operation is that the amount of data that can be input or output at any one time is limited to the amount of free memory available in the computer.

In double-buffered operations, the data buffer is configured as a circular buffer. For input operations, the DAQ device fills the circular buffer with data. When the end of the buffer is reached, the board returns to the beginning of the buffer and fills it with data again. This process continues *ad infinitum* until it is interrupted by a hardware error or cleared by a function call.

Double-buffered output operations also use a circular buffer. In this case, however, the DAQ device retrieves data from the circular buffer for output. When the end of the buffer is reached, the board begins retrieving data from the beginning of the buffer again. As for input, the process continues *ad infinitum* until it is interrupted by a hardware error or cleared by a function call.

Unlike single-buffered operations, double-buffered operations reuse the same buffer and are therefore able to input or output an infinite number data points without requiring an infinite amount of memory. However, in order for double buffering to be useful, there must be a means by which to access the data for updating, storage, and processing. The next two sections explain how the data can be accessed for double-buffered input and output operations.

Double-Buffered Input Operations

The data buffer for double-buffered input operations is configured as a circular buffer. In addition, NI-DAQ logically divides the buffer into two equal halves (no actual division exists in the buffer). By dividing the buffer into two halves, NI-DAQ can coordinate user access to the data buffer with the DAQ device. The coordination scheme is simple—NI-DAQ copies data from the circular buffer in sequential halves to a transfer buffer you provide. You can process or store the data in the transfer buffer however you choose.

Figure 5-1 illustrates a series of sequential data transfers.

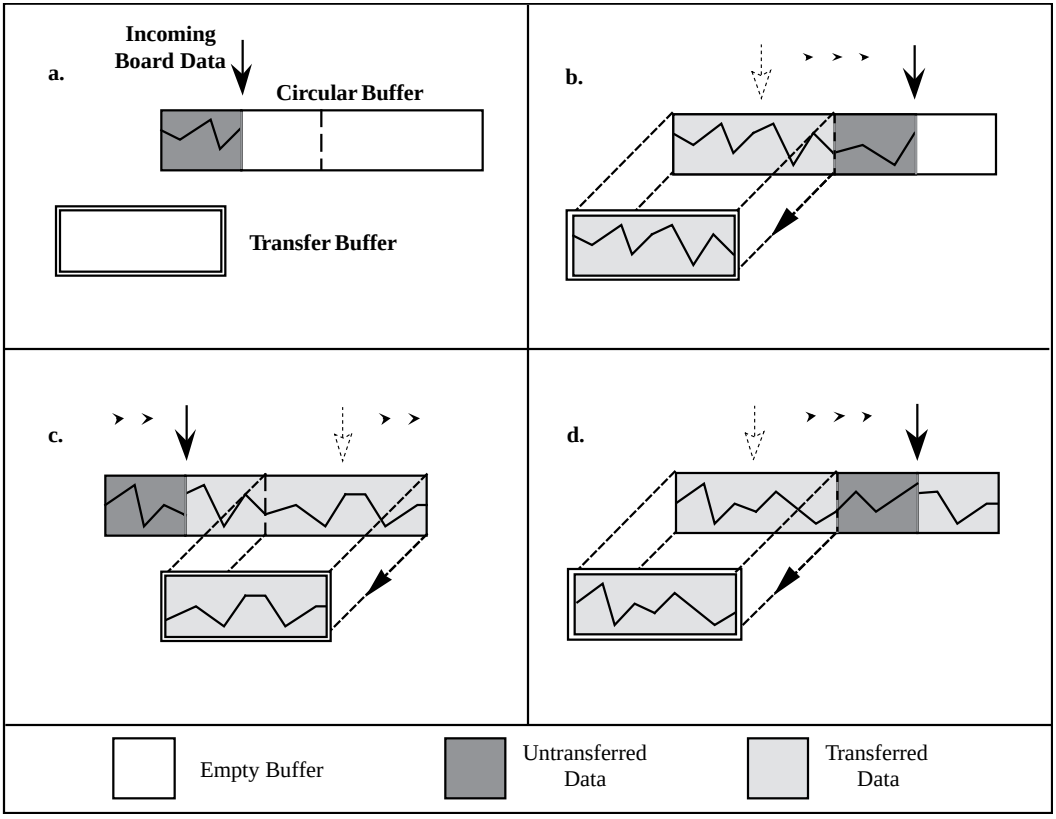


Figure 5-1. Double-Buffered Input with Sequential Data Transfers

The double-buffered input operation begins when the DAQ device starts writing data into the first half of the circular buffer (Figure 5-1a). After the board begins writing to the second half of the circular buffer, NI-DAQ can copy the data from the first half into the transfer buffer (Figure 5-1b). You can then store the data in the transfer block to disk or process it according to the needs of your application. After the input board has filled the second half of the circular buffer, the board returns to the first half buffer and overwrites the old data. NI-DAQ can now copy the second half of the circular buffer to the transfer buffer (Figure 5-1c). The data in the transfer buffer is again available for use by your application. The process can be repeated endlessly to provide a continuous stream of data to your application. You will notice that

Figure 5-1d is equivalent to the step in Figure 5-1b and is the start of a two-step cycle.

Problem Situations

The double-buffered coordination scheme is not flawless. An application might experience two possible problems with double-buffered input. The first is the possibility of the DAQ device overwriting data before NI-DAQ has copied it to the transfer buffer. This situation is illustrated by Figure 5-2.

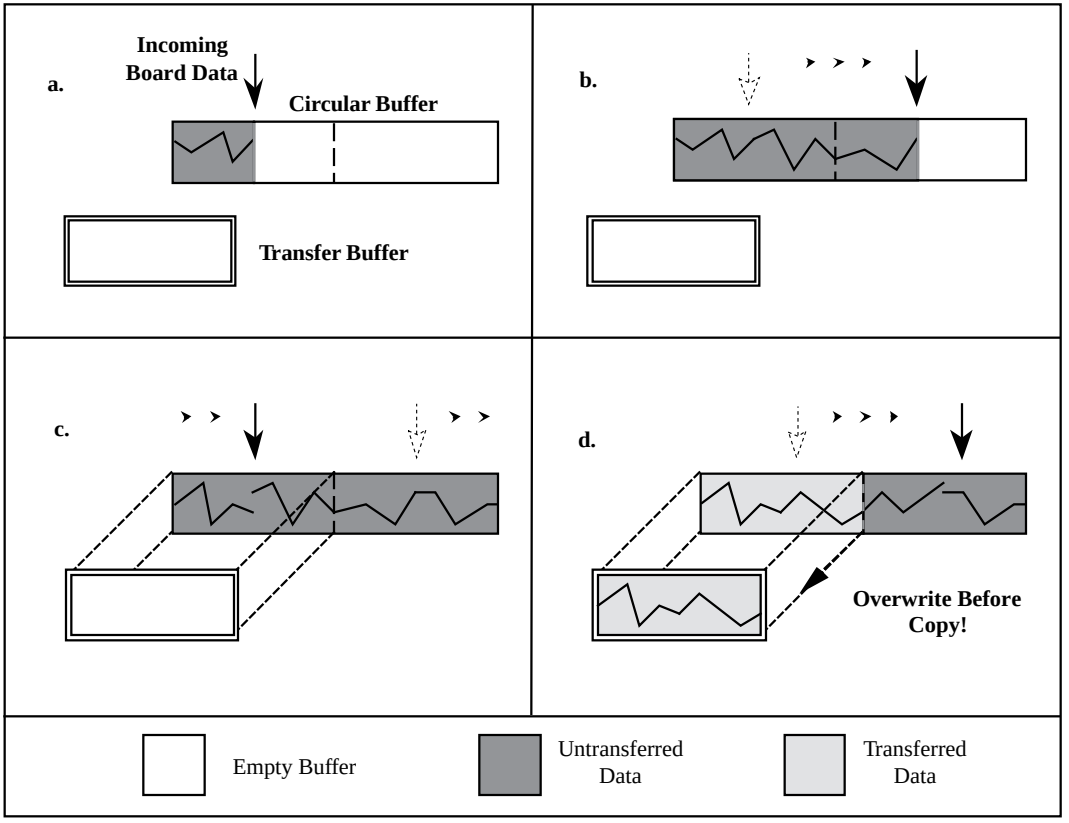


Figure 5-2. Double-Buffered Input with an Overwrite Before Copy

Notice in Figure 5-2b, NI-DAQ has missed the opportunity to copy data from the first half of the circular buffer to the transfer buffer while the DAQ device is writing data to the second half. As a result, the DAQ device begins to overwrite the data in the first half of the circular buffer

before NI-DAQ has copied it to the transfer buffer (Figure 5-2c). To guarantee uncorrupted data, NI-DAQ is forced to wait until the board finishes overwriting data in the first half before copying the data into the transfer buffer. After the board has begun to write to the second half, NI-DAQ copies the data from the first half of the circular buffer to the transfer buffer (Figure 5-2d).

For the previously described situation, NI-DAQ returns an overwrite before copy warning (**12 overWriteBeforeCopy**). This warning indicates that the data in the transfer buffer is valid, but some earlier input data has been lost. Subsequent transfers will not return the warning as long as they keep pace with the DAQ device as in Figure 5-1.

The second potential problem occurs when an input board overwrites data that NI-DAQ is simultaneously copying to the transfer buffer. NI-DAQ returns an overwrite error (**-97 overWriteErr**) when this occurs. The situation is presented in Figure 5-3.

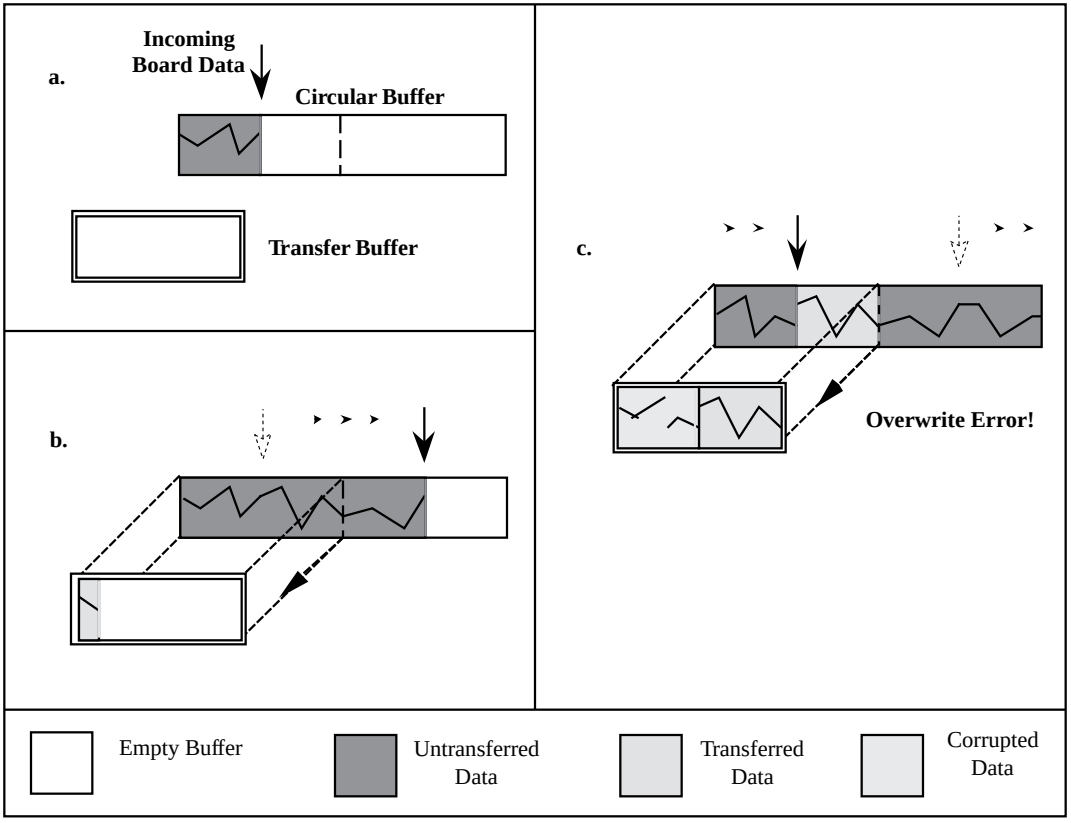


Figure 5-3. Double-Buffered Input with an Overwrite

In Figure 5-3b, NI-DAQ has started to copy data from the first half of the circular buffer into the transfer buffer. However, NI-DAQ is unable to copy the entire half before the DAQ device begins overwriting data in the first half buffer (Figure 5-3c). Consequently, data copied into the transfer buffer may be corrupted; that is, it may contain both old and new data points. Future transfers will execute normally as long as neither of the problem conditions occur again.

Double-Buffered Output Operations

Double-buffered output operations are similar to input operations. The circular buffer is again logically divided into two halves. By dividing the buffer into two halves, NI-DAQ can coordinate user access to the

data buffer with the DAQ device. The coordination scheme is simple—NI-DAQ copies data from a transfer buffer you provide to the circular buffer in sequential halves. The data in the transfer buffer can be updated between transfers.

Figure 5-4 illustrates a series of sequential data transfers.

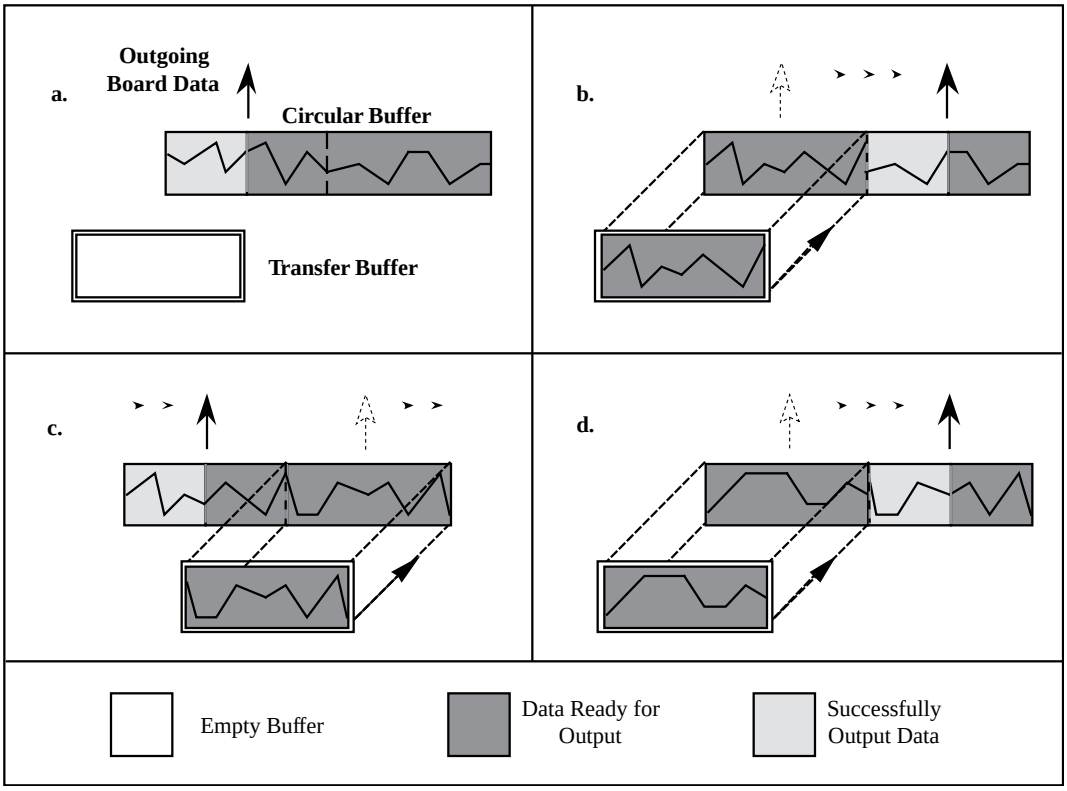


Figure 5-4. Double-Buffered Output with Sequential Data Transfers

The double-buffered output operation begins when the output board begins outputting data from the first half of the circular buffer (Figure 5-4a). After the board begins retrieving data from the second half of the circular buffer, NI-DAQ can copy the prepared data from the transfer buffer to the first half of the circular buffer (Figure 5-4b). The data in the transfer buffer can then be updated with new data by your application. After the output board has finished with the second half of the circular buffer, the board returns to the first half buffer and begins outputting updated data from the first half. NI-DAQ can now copy the

transfer buffer to the second half of the circular buffer (Figure 5-4c). The data in the transfer buffer is again available for update by your application. The process can be repeated endlessly to provide a continuous stream of output data from your application. You will notice that Figure 5-4d is equivalent to the step in Figure 5-4b and is the start of a two-step cycle.

Problem Situations

Like double-buffered input, double-buffered output can experience two potential problems. The first is the possibility of the output board retrieving and outputting the same data before NI-DAQ has updated the circular buffer with new data from the transfer buffer. This situation is illustrated by Figure 5-5.

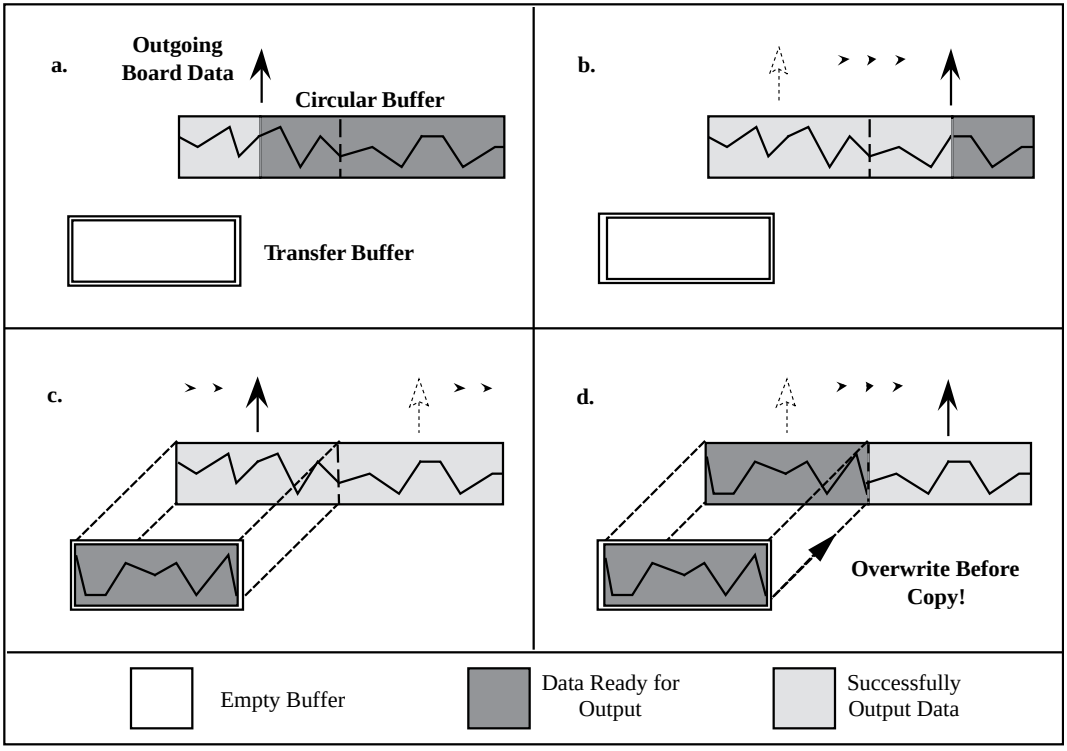


Figure 5-5. Double-Buffered Output with an Overwrite Before Copy

Notice in Figure 5-5b, NI-DAQ has missed the opportunity to copy data from the transfer buffer to the first half of the circular buffer while

the output board is retrieving data from the second half. As a result, the board begins to output the original data in the first half of the circular buffer before NI-DAQ has updated it with data from the transfer buffer (Figure 5-5c). To guarantee uncorrupted output data, NI-DAQ is forced to wait until the board finishes retrieving data from the first half before copying the data from the transfer buffer. After the board has begun to output the second half, NI-DAQ copies the data from the transfer buffer to the first half of the circular buffer (Figure 5-5d).

For this situation, NI-DAQ returns an overwrite before copy warning (**12 overWriteBeforeCopy**). This warning indicates that the board has output old data but the data was uncorrupted during output. Subsequent transfers will not return the warning as long as they keep pace with the output board as in Figure 5-4.

The second potential problem is when an output board retrieves data that NI-DAQ is simultaneously overwriting with data from the transfer buffer. NI-DAQ returns an overwrite error (**-97 overWriteErr**) when this occurs. The situation is presented in Figure 5-6.

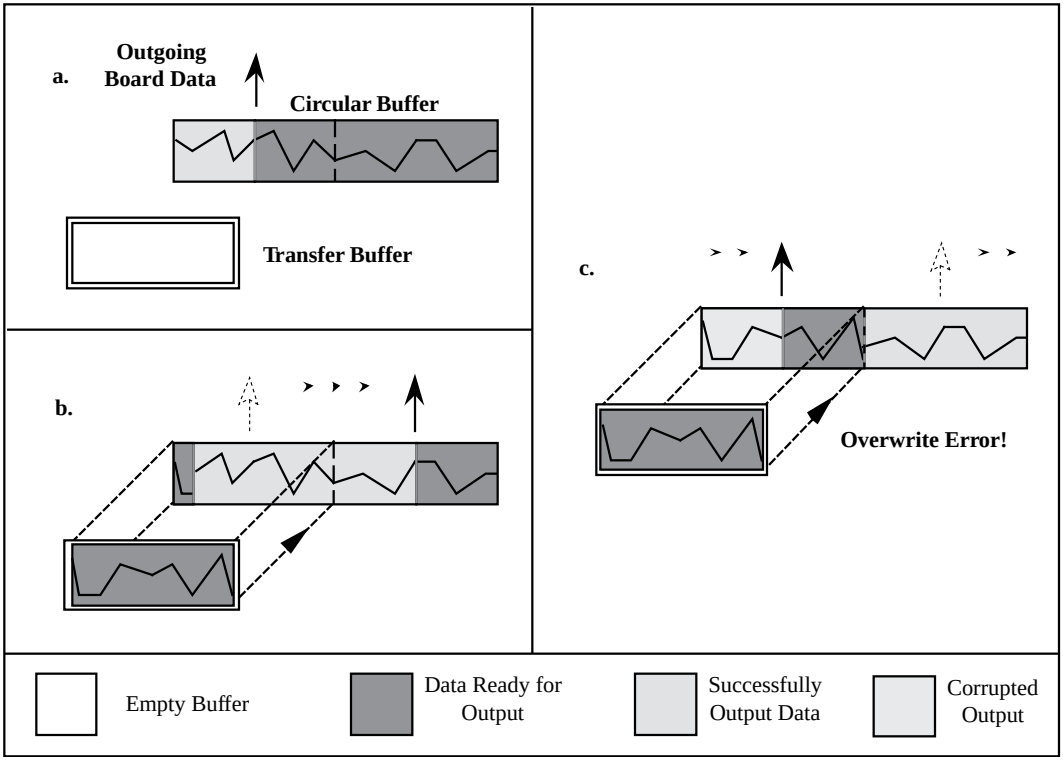


Figure 5-6. Double-Buffered Output with an Overwrite

In Figure 5-6b, NI-DAQ has started to copy data from the transfer buffer to the first half of the circular buffer. However, NI-DAQ is unable to copy all of the data before the output board begins retrieving data from the first half (Figure 5-6c). Consequently, data output by the board may be corrupted; that is, it may contain both old and new data points. Future transfers will execute normally as long as neither of these problem conditions occur again.

Double-Buffered Functions

Double-buffered functions exist for analog input (DAQ), analog output (WFM), and digital input and output (DIG) operations. The functions and the order in which your application should call them is nearly identical for all four operations. This section explains what each of the functions do and the order in which you should call them.

DB_Config Functions

The DB_Config functions enable and disable double buffering for input and output operations, and allow you to select double-buffering options if any are available.

The configuration functions are as follows:

- DAQ_DB_Config
- WFM_DB_Config
- DIG_DB_Config

For analog input operations, call DAQ_DB_Config prior to calling DAQ_Start or a SCAN_Start to enable or disable double buffering. For waveform operations, call WFM_DB_Config prior to calling WFM_Load to enable or disable double buffering. For digital block input and output operations, call DIG_DB_Config prior to calling DIG_Block_In or DIG_Block_Out to enable or disable double buffering.

DB_Transfer and DB_StrTransfer Functions

After a double-buffered operation has been started, the DB_Transfer functions transfer data to or from a circular buffer. The direction of the transfer depends on the direction of the double-buffered operations. Along with copying data, the DB_Transfer functions also check for errors that can occur during the transfer.

For input operations, DB_Transfer copies data from alternating halves of the circular input buffer to the buffer passed to the function (that is, the transfer buffer). For output operations, DB_Transfer copies data from the buffer passed to the function to alternating halves of the circular output buffer. The function may return an overwrite before copy warning (**12 overWriteBeforeCopy**) or an overwrite error (**-97 overWriteErr**) if a problem occurs during the transfer.

☞ **Note:** ***Waveform transfer functions do not detect overwrite before copy or overwrite errors.***

The DB_Transfer functions are synchronous for both input and output operations. In other words, when your application calls these functions, NI-DAQ does not return control to your application until the transfer is complete. As a result, your application may crash if NI-DAQ cannot complete the transfer. To avoid this situation, call the Timeout_Config function prior to starting a double-buffered

operation. The timeout configuration function sets the maximum time allowed to complete a synchronous function call for a board.

The transfer functions are as follows:

- DAQ_DB_Transfer
- WFM_DB_Transfer
- DIG_DB_Transfer

For analog input operations, call DAQ_DB_Transfer after starting a double-buffered analog acquisition to perform a double-buffered transfer. For waveform operations, call WFM_DB_Transfer after starting a double-buffered waveform generation to perform a double-buffered transfer. For digital block input and output operations, call DIG_DB_Transfer after starting a double-buffered digital operation to perform a double-buffered transfer.

The DB_StrTransfer functions are identical in functionality to the DB_Transfer. The DB_StrTransfer functions are for BASIC language applications. The functions operate on strings, making it possible to write the data to a disk file for double-buffered input, or to retrieve data from disk for double-buffered output.

DB_HalfReady Functions

With the DB_HalfReady functions, applications can avoid the delay that can occur when calling the DB_Transfer or DB_StrTransfer functions. When you call either of the transfer functions, NI-DAQ waits until the transfer to or from the circular buffer can be made; that is, the DAQ device is operating on the opposite half of the circular buffer.

The DB_HalfReady functions check if a DB_Transfer can be completed immediately. If the call to DB_HalfReady indicates a transfer cannot be made, your application can do other work and try again later.

The HalfReady functions are as follows:

- DAQ_DB_HalfReady
- WFM_DB_HalfReady
- DIG_DB_HalfReady

For analog input operations, call `DAQ_DB_HalfReady`, after starting a double-buffered analog acquisition but prior to calling `DAQ_DB_Transfer`, to check the transfer status of the operation.

For analog output problems, call `WFM_DB_HalfReady`, after starting a double-buffered waveform generation but prior to calling `WFM_DB_Transfer`, to check the transfer status of the operation.

For digital block input and output operations, call `DIG_DB_HalfReady`, after starting a double-buffered digital operation but prior to calling `DIG_DB_Transfer`, to check the transfer status of the operation.

Conclusion

Double buffering is a data acquisition software technique for continuously inputting or outputting large amounts of data with limited available system memory. However, double buffering may not be practical for high-speed input or output applications. The throughput of a double-buffered operation is typically limited by the ability of the CPU to process the data within a given period of time. Specifically, data must be processed by the application at least as fast as the rate at which the board is writing or reading data. For many applications, this requirement depends on the speed and efficiency of the computer system and programming language.

Appendix

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: (44) 635 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 the following numbers:

(512) 418-1111 or (800) 329-7177

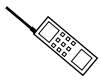
E-Mail Support (currently U.S. only)

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
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

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.



	Fax	Telephone
Australia	(03) 879 9422	(03) 879 9179
Austria	(0662) 435986	(0662) 437010-19
Belgium	02/757.00.20	02/757.03.11
Denmark	45 76 26 00	45 76 71 11
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	02 2637 5019	02 2686 8505
Italy	02/48301892	02/48301915
Japan	(03) 3788-1921	(03) 3788-1923
Korea	02 596-7456	02 596-7455
Mexico	05 202 2544	05 202 2544
Netherlands	03480-33466	03480-30673
Norway	32-848400	32-848600
Singapore	2265886	2265887
Spain	(1) 640 0085	(1) 640 0533
Sweden	08-730 49 70	08-730 43 70
Switzerland	056/20 51 51	056/20 51 55
Taiwan	62 377 1200	62 737 4644
U.K.	1635 523545	1635 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:_____

_____

_____

_____

_____

DAQ 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/CVI 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® User Manual for PC Compatibles, Version 4.8*

Edition Date: May 1995

Part Number: 320498C-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, MS 53-02
Austin, TX 78730-5039

Fax to: Technical Publications
National Instruments Corporation
MS 53-02
(512) 794-5678

Glossary

Prefix	Meaning	Value
μ-	micro-	10^{-6}
m-	milli-	10^{-3}
k-	kilo-	10^{3}
M-	mega-	10^{6}

Numbers/Symbols

α	temperature coefficient at T = 0° C
ß	coefficient
∂	coefficient
Ɛ	strain
Ω	ohm
°	degree
%	percent
+	plus
-	minus
±	plus or minus

A

A/D	analog-to-digital
AC	alternating current

ACK	acknowledge
ADC	A/D converter. An electronic device, often an integrated circuit, that converts an analog voltage to a digital number.
ADC resolution	The resolution of the ADC, which is measured in bits. An ADC with 16 bits has a higher resolution, and thus a higher degree of accuracy, than a 12-bit ADC.
ADF	adapter description file
AI	analog input
AMD	Advanced Micro Devices
analog trigger	A trigger that occurs at a user-selected point on an incoming analog signal. Triggering can be set to occur at a specific level on either an increasing or a decreasing signal (positive or negative slope). Analog triggering can be implemented either in software or in hardware. When implemented in software (LabVIEW), all data is collected, transferred into system memory, and analyzed for the trigger condition. When analog triggering is implemented in hardware, no data is transferred to system memory until the trigger condition has occurred.
API	application programming interface
asynchronous	(1) Hardware—A property of an event that occurs at an arbitrary time, without synchronization to a reference clock. (2) Software—A property of a function that begins an operation and returns prior to the completion or termination of the operation.

B

background acquisition	Data is acquired by a DAQ system while another program or processing routine is running without apparent interruption.
base address	A memory address that serves as the starting address for programmable registers. All other addresses are located by adding to the base address.
BCD	binary-coded decimal
BIOS	basic input/output system
bipolar	A signal range that includes both positive and negative values (for example, -5 V to +5 V).
bit	One binary digit, either 0 or 1.

block-mode	A high-speed data transfer in which the address of the data is sent followed by a specified number of back-to-back data words.
bus	The group of conductors that interconnect individual circuitry in a computer. Typically, a bus is the expansion vehicle to which I/O or other devices are connected. Examples of PC buses are the AT bus, NuBus, Micro Channel, and EISA bus.
byte	Eight related bits of data, an 8-bit binary number. Also used to denote the amount of memory required to store one byte of data.

C

C	Celsius
CI	computing index
cold-junction compensation	A method of compensating for inaccuracies in thermocouple circuits.
compiler	A software utility that converts a source program in a high-level programming language, such as BASIC, C ,or Pascal, into an object or compiled program in machine language. Compiled programs run 10 to 1,000 times faster than interpreted programs.
conversion time	The time required, in an analog input or output system, from the moment a channel is interrogated (such as with a read instruction) to the moment that accurate data is available.
counter/timer	A circuit that counts external pulses or clock pulses (timing).
coupling	The manner in which a signal is connected from one location to another.
CPU	central processing unit

D

D/A	digital-to-analog
DAC	D/A converter. An electronic device, often an integrated circuit, that converts a digital number into a corresponding analog voltage or current.

DAQ	Data acquisition. (1) Collecting and measuring electrical signals from sensors, transducers, and test probes or fixtures and inputting them to a computer for processing. (2) Collecting and measuring the same kinds of electrical signals with A/D and/or DIO boards plugged into a PC, and possibly generating control signals with D/A and/or DIO boards in the same PC.
DC	direct current
device	Device is used to refer to a DAQ device inside your computer or attached directly to your computer via a parallel port. Plug-in boards, PCMCIA cards, and devices such as the DAQPad-1200, which connects to your computer parallel port, are all examples of DAQ devices. SCXI modules are distinct from devices, with the exception of the SCXI-1200, which is a hybrid.
differential input	An analog input consisting of two terminals, both of which are isolated from computer ground, whose difference is measured.
digital port	*See* port.
DIN	Deutsche Industrie Norme
DIO	digital I/O
DLL	Dynamic-link library. A software module in Microsoft Windows containing executable code and data that can be called or used by Windows applications or other DLLs. Functions and data in a DLL are loaded and linked at run time when they are referenced by a Windows application or other DLLs.
DMA	Direct memory access. A method by which data can be transferred to/from computer memory from/to a device or memory on the bus while the processor does something else. DMA is the fastest method of transferring data to/from computer memory.
DOS	Disk Operating System
driver	Software that controls a specific hardware device such as a DAQ board or a GPIB interface board.
DSP	digital signal processing

E

EEPROM	Electronically erasable programmable read-only memory. ROM that can be erased with an electrical signal and reprogrammed.

EGA	enhanced graphics adapter
EISA	Extended Industry Standard Architecture
external trigger	A voltage pulse from an external source that triggers an event such as A/D conversion.

F

FIFO	A first-in first-out memory buffer; the first data stored is the first data sent to the acceptor. FIFOs are often used on DAQ devices to temporarily store incoming or outgoing data until that data can be retrieved or output. For example, an analog input FIFO stores the results of A/D conversions until the data can be retrieved into system memory, a process that requires the servicing of interrupts and often the programming of the DMA controller. This process can take several milliseconds in some cases. During this time, data accumulates in the FIFO for future retrieval. With a larger FIFO, longer latencies can be tolerated. In the case of analog output, a FIFO permits faster update rates, because the waveform data can be stored on the FIFO ahead of time. This again reduces the effect of latencies associated with getting the data from system memory to the DAQ device.

G

gain	The factor by which a signal is amplified, sometimes expressed in decibels.
group	A collection of digital ports, combined to form a larger entity for digital input and/or output.

H

Hz	hertz

I

IBM	International Business Machines
ID	identification
IDE	Integrated Development Environment
IEEE	Institute of Electrical and Electronics Engineers

interrupt	A computer signal indicating that the CPU should suspend its current task to service a designated activity.
I/O	input/output
IRQ	interrupt request
ISA	Industry Standard Architecture

K

kS	1,000 samples
Kword	1,024 words of memory

L

LED	light-emitting diode
library	A file containing compiled object modules, each comprised of one of more functions, that can be linked to other object modules that make use of these functions. `NIDAQMSC.LIB` is a library that contains NI-DAQ functions. The NI-DAQ function set is broken down into object modules so that only the object modules that are relevant to your application are linked in, while those object modules that are not relevant are not linked.
linker	A software utility that combines object modules (created by a compiler) and libraries, which are collections of object modules, into an executable program.
LSB	least significant bit

M

MB	megabytes of memory
MIO	multifunction I/O
MS	million samples
mux	Multiplexer; a switching device with multiple inputs that sequentially connects each of its inputs to its output, typically at high speeds, in order to measure several signals with a single analog input channel.

N

NBS	National Bureau of Standards
NC	Normally Closed
NIVDMAD	National Instruments Virtual DMA Driver
NIVISRD	National Instruments Virtual Interrupt Service Routine Driver. See the NIVISRD entry in the *Index* for information about the National Instruments Virtual Interrupt Service Routine Driver.

O

output settling time	The amount of time required for the analog output voltage to reach its final value within specified limits.

P

paging	A technique used for extending the address range of a device to point into a larger address space.
PC	personal computer
port	A digital port, consisting of four or eight lines of digital input and/or output.
postriggering	The technique used on a DAQ board to acquire a programmed number of samples after trigger conditions are met.
pretriggering	The technique used on a DAQ board to keep a continuous buffer filled with data, so that when the trigger conditions are met, the sample includes the data leading up to the trigger condition.
programmed I/O	The standard method a CPU uses to access an I/O device—each byte of data is read or written by the CPU.
pts	points

R

RAM	random-access memory
REQ	request

resolution	The smallest signal increment that can be detected by a measurement system. Resolution can be expressed in bits, in proportions, or in percent of full scale. For example, a system has 12-bit resolution, one part in 4,096 resolution, and 0.0244 percent of full scale.
ROM	read-only memory
RTD	Resistance temperature detector. A metallic probe that measures temperature based upon its coefficient of resistivity.
RTSI	Real-Time System Integration (bus). The National Instruments timing bus that connects DAQ boards directly, by means of connectors on top of the boards, for precise synchronization of functions.

S

s	seconds
S	samples
Sample-and-Hold (S/H)	A circuit that acquires and stores an analog voltage on a capacitor for a short period of time.
SCXI	Signal Conditioning eXtensions for Instrumentation; the National Instruments product line for conditioning low-level signals within an external chassis near sensors so only high-level signals are sent to DAQ boards in the noisy PC environment.
SDK	Software Development Kit
self-calibrating	A property of a DAQ board that has an extremely stable onboard reference and calibrates its own A/D and D/A circuits without manual adjustments by the user.
Single-Ended (SE) Inputs	An analog input that is measured with respect to a common ground.
S/s	Samples per second; used to express the rate at which a DAQ board samples an analog signal.
software trigger	A programmed event that triggers an event such as data acquisition.
STC	System Timing Controller
synchronous	(1) Hardware—A property of an event that is synchronized to a reference clock.

	(2) Software—A property of a function that begins an operation and returns only when the operation is complete.

T

TC	terminal count
throughput rate	The data, measured in bytes/s, for a given continuous operation, calculated to include software overhead. Throughput Rate = Transfer Rate - Software Overhead Factor.
TPCX	Turbo Pascal compiler
TPU	Turbo Pascal Unit
transfer rate	The rate, measured in bytes/s, at which data is moved from source to destination after software initialization and set up operations; the maximum rate at which the hardware can operate.
TSR	terminate-and-stay resident

U

unipolar	A signal range that is always positive (for example, 0 to +10 V).

V

V	volts
VDC	volts direct current
VDMAD	Virtual DMA Driver. See the NIVDMAD entry in the *Index* for information about the National Instruments Virtual DMA Driver.
VPICD	Virtual Programmable Interrupt Controller Device

X

XMS	extended memory specification

Index

A

B

C

D

E

F

G

H

I

L

M

N

O

P

Q

R

S

T

U

V

W