Click here to comment on this document
via the National Instruments website at
http://www.natinst.com/documentation/daq/

DAQ

NI-DAQ® User Manual for PC Compatibles

Version 5.1

Data Acquisition Software for the PC

July 1997 Edition
Part Number 321644A-01

© Copyright 1991, 1997 National Instruments Corporation. All Rights Reserved.

Internet Support

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

Bulletin Board Support

BBS United States: (512) 794-5422
BBS United Kingdom: 01635 551422
BBS France: 01 48 65 15 59

Fax-on-Demand Support

(512) 418-1111

Telephone Support (U.S.)

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

International Offices

Australia 03 9879 5166, Austria 0662 45 79 90 0, Belgium 02 757 00 20,
Canada (Ontario) 905 785 0085, Canada (Québec) 514 694 8521, Denmark 45 76 26 00,
Finland 09 725 725 11, France 01 48 14 24 24, Germany 089 741 31 30,
Hong Kong 2645 3186, Israel 03 5734815, Italy 02 413091, Japan 03 5472 2970,
Korea 02 596 7456, Mexico 5 520 2635, Netherlands 0348 433466, Norway 32 84 84 00,
Singapore 2265886, Spain 91 640 0085, Sweden 08 730 49 70, Switzerland 056 200 51 51,
Taiwan 02 377 1200, U.K. 01635 523545

National Instruments Corporate Headquarters

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

Important Information

Warranty

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

Under the copyright laws, this publication may not be reproduced or transmitted in any form, electronic or mechanical, including photocopying, recording, storing in an information retrieval system, or translating, in whole or in part, without the prior written consent of National Instruments Corporation.

Trademarks

LabVIEW®, NI-DAQ®, RTSI®, BridgeVIEW™, ComponentWorks™, CVI™, DAQArb™, DAQCard™, DAQ Designer™, DAQPad™, DAQ-PnP™, DAQ-STC™, DAQWare™, NI-DSP™, NI-PGIA™, SCXI™, and VirtualBench™ are trademarks of National Instruments Corporation.

Product and company names listed are trademarks or trade names of their respective companies.

WARNING REGARDING MEDICAL AND CLINICAL USE OF NATIONAL INSTRUMENTS PRODUCTS

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

Table of Contents

About This Manual

How to Use the NI-DAQ Documentation Set xiii
Organization of This Manual xiii
Conventions Used in This Manual xiv
MIO and AI Device Terminology xvi
National Instruments Documentation xviii
Related Documentation xix
Customer Communication xx

Chapter 1
Introduction to NI-DAQ

About the NI-DAQ Software for PC Compatibles 1-1
How to Set up Your DAQ System 1-2
NI-DAQ Overview 1-4
NI-DAQ Hardware Support 1-4
NI-DAQ Language Support 1-6
Device Configuration 1-6
Configuring PC Cards (PCMCIA) 1-7
Configuring PC Cards for Windows NT 4.0 1-8
Configuring PC Cards for Windows 95 1-8
Configuring Parallel Port DAQ Devices 1-8
Configuring Plug and Play Devices 1-9
Configuring Plug and Play Devices in Windows NT 4.0 1-9
ISA Devices 1-9
PCI Devices 1-10
Configuring Plug and Play Devices in Windows 95 1-10
Configuring Non-Plug and Play Devices on ISA (PC AT/XT) or EISA Computers 1-11
Using The NI-DAQ Configuration Utility 1-11

Chapter 2
Fundamentals of Building Windows Applications

The NI-DAQ Libraries 2-1
Creating a Windows Application Using Borland C++ 2-2

Example Programs 2-2
Special Considerations 2-3
Buffer Allocation 2-3
Huge Buffer Access for 16-Bit Programs 2-3
String Passing 2-3
Parameter Passing 2-3
Creating a Windows Application Using Microsoft Visual C++ 2-3
Example Programs 2-5
Special Considerations 2-6
Creating a Windows Application Using Microsoft Visual Basic 2-6
Example Programs 2-7
Special Considerations 2-7
Buffer Allocation 2-7
String Passing 2-8
Parameter Passing 2-8
Using Borland Delphi with NI-DAQ 2-9
NI-DAQ Examples 2-9

Chapter 3
Software Overview

Initialization and General-Configuration Functions 3-2
Software-Calibration and Device-Specific Functions 3-3
Event Message Functions 3-5
Event Messaging Application Hints 3-5
NI-DAQ Events in Visual Basic for Windows 3-6
Visual Basic Custom Controls 3-6
General DAQ Event 3-7
Analog Trigger Event 3-10
Analog Alarm Event 3-11
Using Multiple Controls 3-13
General DAQ Event Example 3-14
Analog Input Function Group 3-16
The One-Shot Analog Input Functions 3-16
The Single-Channel Analog Input Functions 3-16
Single-Channel Analog Input Application Hints 3-18
Data Acquisition Functions 3-20
High-Level DAQ Data Acquisition Functions 3-21
Low-Level DAQ Data Acquisition Functions 3-22
Low-Level Double-Buffered Data Acquisition Functions 3-25
Data Acquisition Application Hints 3-25
Lab and 1200 Devices Counter/Timer Signals 3-25

DAQCard-500/700, 516 Devices, and LPM Device Counter/Timer Signals 3-26
External Multiplexer Support (AMUX-64T) 3-26
Basic Building Blocks 3-26
Building Block 1: Configuration 3-27
Building Block 2: Start 3-28
Building Block 3: Checking 3-31
Building Block 4: Cleaning Up 3-33
Double-Buffered Data Acquisition 3-33
Multirate Scanning 3-34
Analog Output Function Group 3-37
One-Shot Analog Output Functions 3-37
Analog Output Application Hints 3-38
Simple Analog Output Application 3-39
Analog Output with Software Update Application 3-39
Waveform Generation Functions 3-41
High-Level Waveform Generation Functions 3-41
Low-Level Waveform Generation Functions 3-41
Waveform Generation Application Hints 3-43
Basic Waveform Generation Applications 3-43
Staging-Based Arbitrary Waveform Generation (For DAQArb 5411 Devices Only) 3-51
Reference Voltages for Analog Output Devices 3-54
Minimum Update Intervals 3-55
Notes on DMA Waveform Generation with the AT-MIO-16F-5 3-55
Counter Usage 3-56
Restrictions on the Use of a Delay Rate on the AT-MIO-16X and AT-MIO-64F-5 3-58
FIFO Lag Effect on the MIO Series, E Series, AT-AO-6/10, AT-MIO-16X, and AT-MIO-64F-5 3-58
Externally Triggering Your Waveform Generation Operation 3-59
Digital I/O Function Group 3-59
DIO-24, AT-MIO-16D, AT-MIO-16DE-10, DIO-96, and Lab and 1200 Devices Groups 3-62
DIO-32F and DAQDIO 6533 (DIO-32HS) Groups 3-62
Digital I/O Functions 3-63
Group Digital I/O Functions 3-64
Double-Buffered Digital I/O Functions 3-66
Digital I/O Application Hints 3-67
Handshaking Versus No-Handshaking Digital I/O 3-67
Digital Port I/O Applications 3-67

Digital Line I/O Applications 3-69
Digital Group I/O Applications..... 3-71
Digital Group Block I/O Applications 3-72
Digital Double-Buffered Group Block I/O Applications..... 3-74
Pattern Generation I/O with the DIO-32F and DAQDIO 6533 (DIO-32HS) Devices..... 3-77
Double-Buffered I/O 3-78
The Counter/Timer Function Group..... 3-79
Device Support for the Counter/Timer and Interval Counter/Timer Functions 3-79
The General-Purpose Counter/Timer Functions 3-80
Counter/Timer Operation for the CTR Functions 3-81
Programmable Frequency Output Operation..... 3-84
Counter/Timer Application Hints 3-85
Event-Counting Applications..... 3-89
Period and Continuous Pulse-Width Measurement Applications..... 3-92
The Interval Counter/Timer Functions 3-94
Interval Counter/Timer Operation for the ICTR Functions..... 3-94
Interval Counter/Timer Application Hints 3-95
The General-Purpose Counter/Timer Functions 3-96
The General-Purpose Counter/Timer Application Hints..... 3-96
The RTSI Bus Trigger Functions 3-97
The RTSI Bus..... 3-98
MIO-16/16D RTSI Connections 3-99
MIO-F-5/16X RTSI Connections 3-100
E Series Devices RTSI Connections 3-101
AT-AO-6/10 RTSI Connections 3-101
DIO-32F RTSI Connections 3-102
DAQDIO 6533 (DIO-32HS) RTSI Connections 3-102
RTSI Bus Application Hints 3-104
The SCXI Functions 3-105
SCXI Application Hints 3-109
Building Analog Input Applications in Multiplexed Mode..... 3-111
Building Analog Input Applications in Parallel Mode 3-118
SCXI Data Acquisition Rates 3-122
Analog Output Applications 3-124
Digital Applications 3-125
The Transducer Conversion Functions..... 3-125
Transducer Conversion Function Descriptions..... 3-127
RTD_Convert 3-127
RTD_Buf_Convert 3-127
Strain_Convert..... 3-129

Strain_Buf_Convert 3-129
Thermistor_Convert 3-132
Thermistor_Buf_Convert 3-132
Thermocouple_Convert 3-134
Thermocouple_Buf_Convert 3-134

Chapter 4
DMA and Programmed I/O Performance Limitations

Explanation of Programmed I/O and DMA Transfers 4-1
Programmed I/O or DMA 4-2
Using DMA on AT Bus Computers 4-2
Page Boundaries in AT Bus Computers 4-2
Using Physical Memory above 16 MB on ISA Bus Computers 4-4
General Performance Considerations for Windows 4-5
Programmed I/O Performance in Windows 4-5
DMA Performance in Windows 4-5
Buffers Requiring Reprogramming 4-5
Why Reprogramming Limits Performance 4-6
Results of Performance Limitation 4-6
Methods for Eliminating Performance Limitations 4-7

Chapter 5
NI-DAQ Double Buffering

Overview 5-1
Single-Buffered Versus Double-Buffered Data 5-1
Double-Buffered Input Operations 5-2
Problem Situations 5-4
Double-Buffered Output Operations 5-6
Problem Situations 5-7
Double-Buffered Functions 5-9
DB_Config Functions 5-10
DB_Transfer Functions 5-10
DB_HalfReady Functions 5-11
Conclusion 5-12

Customer Communication

Glossary

Figures

Figure 1-1. How to Set up Your DAQ System 1-3

Figure 3-1. Single-Point Analog Reading with Onboard Conversion Timing..... 3-19
Figure 3-2. Single-Point Analog Reading with External Conversion Timing 3-20
Figure 3-3. Buffered Data Acquisition Basic Building Blocks..... 3-27
Figure 3-4. Buffered DAQ Data Acquisition Application Building Block 1, Configuration..... 3-28
Figure 3-5. Buffered Data Acquisition Application Building Block 2, Start, for the MIO and AI Devices..... 3-30
Figure 3-6. Buffered Data Acquisition Application Building Block 2, Start, for the 516 Devices, DAQCard-500/700, Lab and 1200 Devices, and LPM Devices 3-31
Figure 3-7. Buffered Data Acquisition Application Building Block 3, Checking, for the MIO and AI Devices..... 3-32
Figure 3-8. Buffered Data Acquisition Application Building Block 3, Checking, for the 516 Devices, DAQCard-500/700, Lab and 1200 Devices, and LPM Devices..... 3-32
Figure 3-9. Double-Buffered DAQ Data Acquisition Application Building Block 3, Checking 3-34
Figure 3-10. Multirate Scanning 3-36
Figure 3-11. Equivalent Analog Output Calls..... 3-38
Figure 3-12. Simple Analog Output Application..... 3-39
Figure 3-13. Analog Output with Software Updates 3-40
Figure 3-14. Basic Waveform Generation Application 3-45
Figure 3-15. Waveform Generation with Pauses 3-48
Figure 3-16. Double-Buffered Waveform Generation..... 3-50
Figure 3-17. Staging-Based Waveform Generation..... 3-53
Figure 3-18. Basic Port Input or Output Application with Handshaking 3-68
Figure 3-19. Basic Port Input or Output Application without Handshaking 3-68
Figure 3-20. Basic Line Input or Output Application 3-70
Figure 3-21. Simple Digital Group Input or Output Application..... 3-71
Figure 3-22. Digital Block Input or Output Application..... 3-73
Figure 3-23. Double-Buffered Block Operation 3-75
Figure 3-24. Counter Block Diagram..... 3-81
Figure 3-25. Counter Timing and Output Types..... 3-84
Figure 3-26. Event Counting..... 3-86
Figure 3-27. Pulse Generation..... 3-87
Figure 3-28. Simultaneous Counter Operation 3-88
Figure 3-29. Timer Event Counting..... 3-89
Figure 3-30. Pulse-Width Measurement 3-90
Figure 3-31. Frequency Measurement 3-91
Figure 3-32. Continuous Period Measurement 3-93

Figure 3-33. Interval Counter Block Diagram 3-95
Figure 3-34. Basic RTSI Application Calls 3-105
Figure 3-35. General SCXIbus Application 3-110
Figure 3-36. Single-Channel or Software-Scanning Operation Using the SCXI-1100, SCXI-1102, VXI-SC-1102, SCXI-1120, SCXI-1120D, SCXI-1121, SCXI-1122, or SCXI-1141 in Multiplexed Mode 3-112
Figure 3-37. Single-Channel or Software-Scanning Operation Using the SCXI-1140 in Multiplexed Mode 3-114
Figure 3-38. Channel-Scanning Operation Using Modules in Multiplexed Mode 3-116
Figure 3-39. Single-Channel or Software-Scanning Operation Using the SCXI-1140 in Parallel Mode 3-120
Figure 3-40. Channel-Scanning Operation Using the SCXI-1140 in Parallel Mode 3-121
Figure 3-41. Strain Gauge Bridge Configuration 3-131
Figure 3-42. Circuit Diagram of a Thermistor in a Voltage Divider 3-134

Figure 5-1. Double-Buffered Input with Sequential Data Transfers 5-3
Figure 5-2. Double-Buffered Input with an Overwrite Before Copy 5-4
Figure 5-3. Double-Buffered Input with an Overwrite 5-5
Figure 5-4. Double-Buffered Output with Sequential Data Transfers 5-6
Figure 5-5. Double-Buffered Output with an Overwrite Before Copy 5-8
Figure 5-6. Double-Buffered Output with an Overwrite 5-9

Tables

Table 1. MIO and AI Device Classifications xvii

Table 1-1. NI-DAQ Version 5.1 Internal Device Hardware Support for AT, PC, and NEC 1-4
Table 1-2. NI-DAQ Version 5.1 Internal Device Hardware Support for PCMCIA, PCI, PXI, and VXI 1-5
Table 1-3. NI-DAQ External Device Hardware Support 1-5

Table 3-1. General DAQ Event Control Properties 3-8
Table 3-2. Analog Trigger Event Control Properties 3-10
Table 3-3. Analog Alarm Event Control Properties 3-12
Table 3-4. Output Voltages with Unipolar Output Polarity 3-54
Table 3-5. Output Voltages with Bipolar Output Polarity 3-54
Table 3-6. Mapping of a Byte to Digital I/O Lines 3-60
Table 3-7. Legal Group Assignments for DIO-32F Devices 3-63
Table 3-8. MIO-16/16D RTSI Bus Signals 3-99
Table 3-9. MIO-F-5/16X RTSI Bus Signals 3-100
Table 3-10. MIO-16/16D RTSI Bus Signals 3-101
Table 3-11. DIO-32F RTSI Bus Signals 3-102

Table 3-12. DAQDIO 6533 RTSI Bus Signals..... 3-103
Table 3-13. Maximum SCXI Module Settling Times 3-123
Table 3-14. SCXI-1200 Module Settling Rates..... 3-124
Table 3-15. Temperature Error for Thermocouple Inverse Equations 3-136

About This Manual

The *NI-DAQ User Manual for PC Compatibles* is for users of the NI-DAQ software for PC compatibles version 5.1. NI-DAQ software is a powerful application programming interface (API) between your data acquisition (DAQ) application and the National Instruments DAQ devices. Source code for several example applications is included.

How to Use the NI-DAQ Documentation Set

You should begin by reading the NI-DAQ release notes and this manual. Chapter 1, *Introduction to NI-DAQ*, contains a flowchart that illustrates how to set up your DAQ system using either NI-DAQ or LabVIEW.

When you are familiar with the material in this manual, you can begin to use the *NI-DAQ Function Reference Manual for PC Compatibles*. The *NI-DAQ Function Reference Manual for PC Compatibles* is a reference manual that contains detailed descriptions of the NI-DAQ functions. You also can use the Windows help file `NIDAQPC.HLP` located in your NI-DAQ software, 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 set up your DAQ system and configure your DAQ devices.
- Chapter 2, *Fundamentals of Building Windows Applications*, describes the fundamentals of creating NI-DAQ applications in Windows and Windows NT.
- 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.

1102 Series modules	Refers to the SCXI-1102, SCXI-1102B, SCXI-1102C modules and the VXI-SC-1102, VXI-SC-1102B, and VXI-SC-1102C submodules.
12-bit device	These are MIO and AI devices, such as the AT-MIO-16, AT-MIO-64E-3, PXI-6070E, PXI-6040E, and PCI-6071E. See Table 1 for a complete list.
16-bit device	These are MIO and AI devices, such as the AT-MIO-16X, AT-MIO-16XE-50, PCI-6031E, PCI-6032E, and PCI-6033E. See Table 1 for a complete list.
516 device	Refers to the DAQCard-516 and PC-516.
AI device	Refers to analog input devices, such as the NEC-AI-16E-4, PCI-6032E, and PCI-6033E. See Table 1 for a complete list.
Am9513-based device	These MIO devices do *not* have an *E*- in their names. These devices are the AT-MIO-16, AT-MIO-16F-5, AT-MIO-16X, AT-MIO-16D, and AT-MIO-64F-5.
bold	Bold text denotes the names of menus, menu items, parameters, dialog boxes, dialog box buttons or options, icons, windows, Windows 95 tabs or pages, or LEDs.
bold italic	Bold italic text denotes a note, caution, or warning.
DAQArb 5411 device	Refers to arbitrary waveform generator devices, such as the AT-5411 and PCI-5411.
DAQCard-500/700	Refers to the DAQCard-500 and DAQCard-700.
DAQDIO 6533	Refers to the AT-DIO-32HS, PCI-DIO-32HS, DAQCard-6533, and PXI-6533.

DIO-24	Refers to the PC-DIO-24, PC-DIO-24PnP, and DAQCard-DIO-24.
DIO-32F	Refers to the AT-DIO-32F.
DIO-96	Refers to the PC-DIO-96, PC-DIO-96PnP, and PCI-DIO-96.
DIO board	Refers to any DIO-24, DIO-32, 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, DAQPad-MIO-16XE-50, and PCI-6033E.
italic	Italic text denotes emphasis, a cross reference, or an introduction to a key concept. This font also denotes text for which you supply the appropriate word or value, such as in NI-DAQ 5.*x*.
`italic monospace`	Italic text in this font denotes that you must supply the appropriate words or values in the place of these items.
Lab and 1200 device	Refers to the DAQCard-1200, DAQPad-1200, Lab-PC+, Lab-PC-1200, Lab-PC-1200AI, PCI-1200, and SCXI-1200.
Lab and 1200 analog output device	Refers to the DAQCard-1200, DAQPad-1200, Lab-PC+, Lab-PC-1200, PCI-1200, and SCXI-1200.
LPM device	Refers to the PC-LPM-16 and PC-LPM-16PnP.
MIO device	Refers to multifunction I/O devices, such as the AT-MIO-16, NEC-MIO-16E-4, PXI-6040E, PXI-6070E, PCI-6031E, and PCI-6071E (see the *MIO and AI Device Terminology* section in this chapter).
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, and PCI-MIO-16XE-50.
MIO-64	Refers to the AT-MIO-64F-5, AT-MIO-64E-4, PCI-6031E, PCI-6071E, VXI-MIO-64E-1, and VXI-MIO-64XE-10.
`monospace`	Text in this font denotes text or characters that you should literally enter from the keyboard, sections of code, programming examples, and syntax examples. This font also is used for the proper names of disk drives, paths, directories, programs, subprograms, subroutines, device names, functions, operations, variables, filenames, and extensions, and for statements and comments taken from program code.
NI-DAQ	Refers to the NI-DAQ software for PC compatibles, unless otherwise noted.
PC	Refers to the IBM PC/XT, IBM PC AT, and compatible computers.

PCI Series	Refers to the National Instruments products that use the high-performance expansion bus architecture originally developed by Intel to replace ISA and EISA.
Remote SCXI	Refers to an SCXI configuration where either an SCXI-2000 chassis or an SCXI-2400 remote communications module is connected to the serial port of the PC.
SCXI-1102	SCXI-1102 refers to the SCXI-1102, SCXI-1102B, and SCXI-1102C devices.
SCXI-1120	SCXI-1120 refers to only the SCXI-1120.
SCXI-1120D	SCXI-1120D refers to only the SCXI-1120D.
SCXI-1120/D	SCXI-1120/D refers to both the SCXI-1120D and the SCXI-1120.
SCXI analog input module	Refers to the SCXI-1100, SCXI-1102, SCXI-1120, SCXI-1120D, SCXI-1121, SCXI-1122, SCXI-1140, and SCXI-1141.
SCXI analog output module	Refers to the SCXI-1124.
SCXI chassis	Refers to the SCXI-1000, SCXI-1000DC, SCXI-1001, and SCXI-2000.
SCXI communication module	Refers to the SCXI-2400.
SCXI digital module	Refers to the SCXI-1160, SCXI-1161, SCXI-1162, SCXI-1162HV, SCXI-1163, and SCXI-1163R.
SCXI DAQ module	Refers to the SCXI-1200.

The *Glossary* lists abbreviations, acronyms, metric prefixes, mnemonics, symbols, and terms.

MIO and AI Device Terminology

This manual uses generic terms to describe groups of devices whenever possible. The generic terms for the MIO and AI devices are based on the number of bits, the platform, the functionality, and the series name of the devices. For example, *16-bit, E Series devices* refers to the AT-MIO-16XE-50, DAQPad-MIO-16XE-50, NEC-MIO-16XE-50, AT-MIO-16XE-10, VXI-MIO-64XE-10, PCI-MIO-16XE-50, PCI-MIO-16XE-10, PCI-6031E (MIO-64XE-10), PCI-6033E (AI-64XE-10), and PCI-6071E (MIO-64E-1). 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.

Table 1. MIO and AI Device Classifications

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

Table 1. MIO and AI Device Classifications

Device	Number of SE Channels	Bit	Type	Functionality	Series
PCI-6031E (MIO-64XE-10)	64	16-bit	PCI	MIO	E Series
PCI-6032E (AI-16XE-10)	16	16-bit	PCI	AI	E Series
PCI-6033E (AI-64XE-10)	64	16-bit	PCI	AI	E Series
PCI-6071E (MIO-64E-1)	64	12-bit	PCI	MIO	E Series
PCI-MIO-16E-1	16	12-bit	PCI	MIO	E Series
PCI-MIO-16E-4	16	12-bit	PCI	MIO	E Series
PCI-MIO-16XE-10	16	16-bit	PCI	MIO	E Series
PCI-MIO-16XE-50	16	16-bit	PCI	MIO	E Series
PXI-6040E	16	12-bit	PXI	MIO	E Series
PXI-6070E	16	12-bit	PXI	MIO	E Series
VXI-MIO-64E-1	64	12-bit	VXI	MIO	E Series
VXI-MIO-64XE-10	64	16-bit	VXI	MIO	E Series

National Instruments Documentation

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 documentation—Examples of software documentation you might have are the National Instruments application software—LabVIEW, LabWindows®/CVI, ComponentWorks, BridgeVIEW, and VirtualBench—and the NI-DAQ documentation. After you have set up your hardware system, use either the application software or the NI-DAQ documentation to help you write your application. If you have a large and complicated system, it is worthwhile to look through the software documentation before you configure your hardware.

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

☞ **Note:** ***Only NI-DAQ for PC compatibles versions 5.0 and earlier support Windows 3.x.***

- 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 User Manual*—This manual contains maintenance information on the chassis, installation instructions, and information about making custom modules.

Related Documentation

For detailed hardware information, refer to the user manual included with each device. 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.

Chapter 1

Introduction to NI-DAQ

This chapter describes how to set up your DAQ system and configure your DAQ devices.

About the NI-DAQ Software for PC Compatibles

Thank you for buying a National Instruments DAQ device, 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 acquire and process data simultaneously. 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 use several functions and several devices simultaneously. The Resource Manager prevents multiple-board contention over DMA channels, interrupt levels, and RTSI channels.

NI-DAQ can send *event-driven messages* to Windows or Windows NT applications each time a user-specified event occurs. 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.

How to Set up Your DAQ System

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

If you are accessing the NI-DAQ device drivers through LabVIEW, you should read the NI-DAQ release notes and then use your *LabVIEW Data Acquisition VI Reference Manual* to help you get started using the data acquisition VIs in LabVIEW.

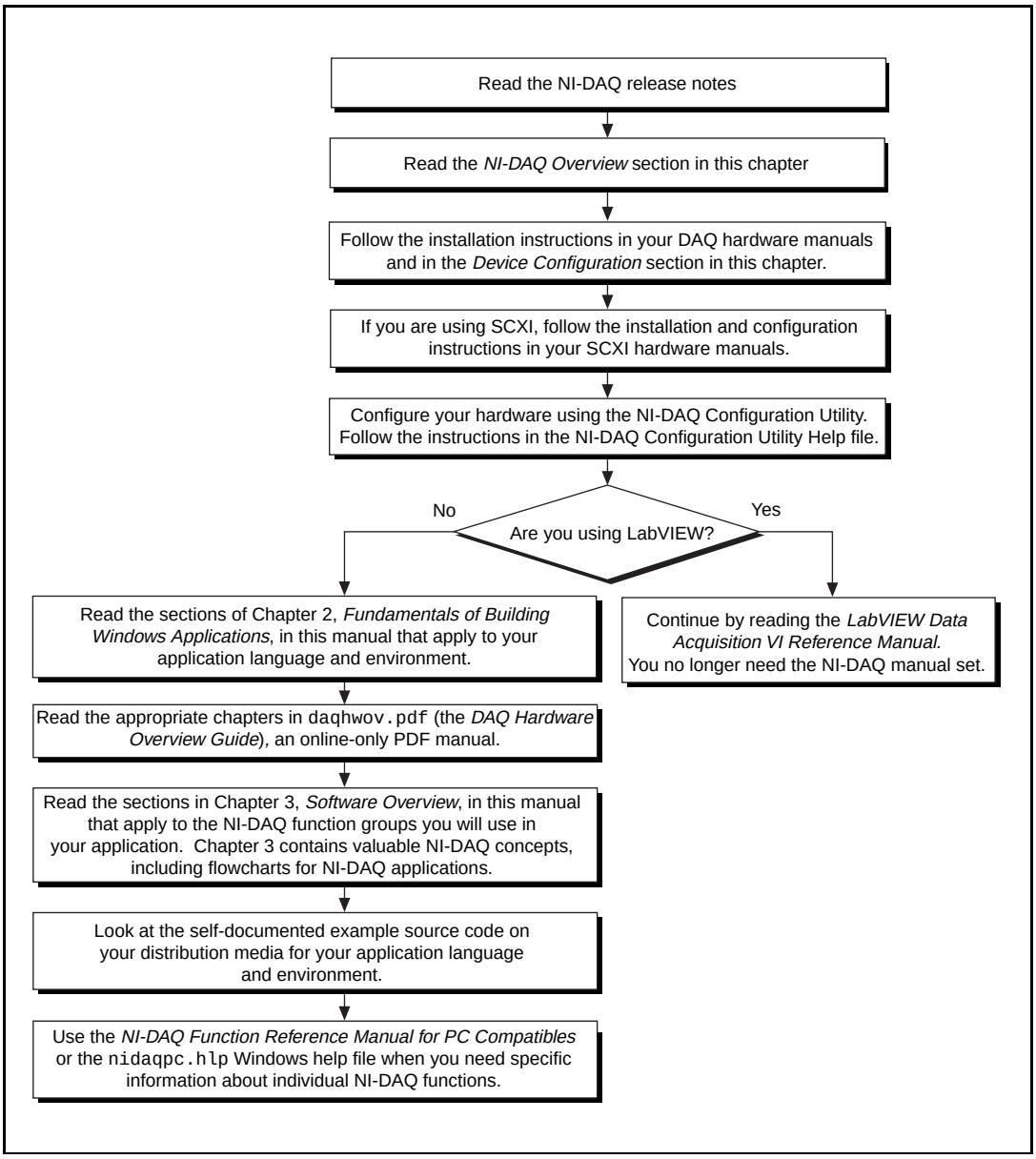


Figure 1-1. How to Set up Your DAQ System

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 Tables 1-1 through 1-3.

Table 1-1. NI-DAQ Version 5.1 Internal Device Hardware Support for AT, PC, and NEC

AT	PC	NEC
AT-AI-16XE-10	Lab-PC+	NEC-AI-16E-4
AT-AO-6/10	PC-516	NEC-AI-16XE-50
AT-DIO-32F	PC-AO-2DC	NEC-MIO-16E-4
AT-MIO-16	PC-DIO-24	NEC-MIO-16XE-50
AT-MIO-16D	PC-DIO-24PnP	
AT-MIO-16DE-10	PC-DIO-96	
AT-MIO-16E-1	PC-DIO-96PnP	
AT-MIO-16E-2	PC-LPM-16	
AT-MIO-16E-10	PC-LPM-16PnP	
AT-MIO-16F-5	PC-OPDIO-16	
AT-MIO-16X	PC-TIO-10	
AT-MIO-16XE-10	Lab-PC-1200	
AT-MIO-16XE-50	Lab-PC-1200AI	
AT-MIO-64E-3		
AT-DIO-32HS		
AT-5411		

Table 1-2. NI-DAQ Version 5.1 Internal Device Hardware Support for PCMCIA, PCI, PXI, and VXI

PCMCIA	PCI	PXI	VXI
DAQCard-500	PCI-MIO-16E-1	PXI-6040E (MIO-16E-4)	VXI-MIO-64E-1
DAQCard-516	PCI-MIO-16E-4	PXI-6032E (AI-16XE-10)	VXI-MIO-64XE-10
DAQCard-700	PCI-MIO-16XE-10	PXI-6533	VXI-DIO-128
DAQCard-1200	PCI-MIO-16XE-50		VXI-AO-48XDC
DAQCard-AI-16E-4	PCI-DIO-96		
DAQCard-AI-16XE-50	PCI-1200		
DAQCard-AO-2DC	PCI-6031E (MIO-64XE-10)		
DAQCard-DIO-24	PCI-6032E (AI-16XE-10)		
DAQCard-6533	PCI-6033E (AI-64XE-10)		
	PCI-6071E (MIO-64E-1)		
	PCI-DIO-32HS		
	PCI-5411		

Table 1-3. NI-DAQ External Device Hardware Support

SCXI		Other Devices
SCXI-1000	SCXI-1122	AMUX-64T
SCXI-1000DC	SCXI-1124	DAQPad-1200[1]
VXI-SC-1000	SCXI-1140	DAQPad-MIO-16XE-50[1]
SCXI-1001	SCXI-1141	SC-2040
SCXI-1100	VXI-SC-1150	SC-2042-RTD
SCXI-1102	SCXI-1160	SC-2043-SG
SCXI-1102B	SCXI-1161	
SCXI-1102C	SCXI-1162	
VXI-SC-1102	SCXI-1162HV	
VXI-SC-1102B	SCXI-1163	
VXI-SC-1102C	SCXI-1163R	
SCXI-1120	SCXI-1200[2]	
SCXI-1120D	SCXI-2000	
SCXI-1121	SCXI-2400	

[1] These devices do not work with NEC PC-9800 computers.
[2] This device works with NEC PC-9800 computers only when used with remote SCXI.

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 always is referred to in the manual by its name.

NI-DAQ Language Support

NI-DAQ supplies header files, examples, and instructions on how to use an Integrated Development Environment (IDE) for one of the following languages under Windows 95/NT:

- Microsoft Visual C++ 2.*x*, 4.*x*
- Visual Basic 4.0 (32-bit)
- Borland C++ 4.5, 5.*x*

NI-DAQ also provides an NI-DAQ function prototype file for use with Borland Delphi 2.*x* (32-bit).

Most of the files on the release media 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.WRI` file.

Device 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 the parallel port of your computer. 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 procedure ensures that your DAQ devices work properly and coexist with other peripherals in your system such as serial ports and parallel ports.

In most cases you follow the same general steps:

1. Install your application software (LabVIEW or CVI, for example).
2. Install your NI-DAQ driver software and your appropriate support files.

3. Install your data acquisition hardware device.
4. Configure your device using the NI-DAQ Configuration Utility.

The NI-DAQ driver software that ships with your hardware is the latest version of NI-DAQ at the time your hardware was shipped. In some cases, the application software will ship and install a version of the NI-DAQ driver software, which might not be as recent as the version of NI-DAQ that you received with your hardware. In such cases, it is a good idea to install the latest version of NI-DAQ after you have installed the application software. In ALL cases, the NI-DAQ software should be installed prior to the installation of the data acquisition hardware.

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 interrupt request (IRQ) levels and/or DMA channels. If you have multiple DAQ devices in your system, select different interrupt request levels and DMA channels for each device. You also should select a unique base address for each device if your device has address DIP switches. Make a note of your device interrupt request, 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, Plug and Play devices, and non-Plug and Play devices for ISA (PC AT/XT) and EISA computers. The following sections also assume that you have already installed your application software (LabVIEW or CVI, for example) and that you have already installed the latest version of NI-DAQ.

Configuring PC Cards (PCMCIA)

The configuration procedure for your PC Card depends on which operating system you are using. Follow the instructions below for the respective operating system you are using.

Configuring PC Cards for Windows NT 4.0

Windows NT 4.0 automatically installs the PC Card driver if it detects PC Card slots when you install the Windows NT 4.0 operating system on your machine. To install and configure your PC Card for Windows NT 4.0, follow the instructions below:

1. Read the *Using The NI-DAQ Configuration Utility* section later in this chapter. You can then run the NI-DAQ Configuration Utility to configure your PC Card and assign it available resources (interrupt and base I/O address).
2. Shut down your computer.
3. Insert the PC Card, and then start your computer.
4. Run the NI-DAQ Configuration Utility. Click on the **System** tab and the **Test** button to ensure that the card is responding properly to the selected interrupt and base I/O address settings. If the device is not responding to the selected interrupt or base I/O setting or if your computer freezes, you need to assign different resources and repeat steps 2 and 4.

If you have SCXI hardware, you should read the SCXI installation instructions in your SCXI user manual.

Configuring PC Cards for Windows 95

Windows 95 and PC Cards work very well together because Windows 95 includes Plug and Play capability and standard drivers for PC Card devices. To configure your PC Card for Windows 95, insert the PC Card into your computer. After you have read the *Using The NI-DAQ Configuration Utility* section later in this chapter, run the NI-DAQ Configuration Utility. If you have SCXI hardware, you should read the SCXI installation instructions in your SCXI user manual.

Configuring Parallel Port DAQ Devices

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

If you are using Windows 95, you also must notify the Windows Device Manager that you have a parallel port DAQ device. Refer to the NI-DAQ Configuration Utility help file for detailed instructions.

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

Configuring Plug and Play 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 DAQ devices that support Plug and Play configuration are all E Series devices, all PCI devices, the Lab-PC-1200/AI, PC-516, PC-AO-2DC, PC-OPDIO-16, PC-DIO-24PnP, PC-DIO-96PnP, PC-LPM-16PnP, AT-5411, AT-DIO-32HS, PCI-DIO-32HS, DAQCard-6533, and PXI-6533. All resources including base address, DMA channels, and interrupt request levels on these devices are fully software configurable. No jumpers or DIP switches are needed to configure any of these resources.

Configuring Plug and Play Devices in Windows NT 4.0

You can install either ISA or PCI Plug and Play devices in Windows NT 4.0. Refer to the appropriate section below to learn how to install either kind of these devices.

ISA Devices

If you plan to use ISA Plug and Play devices (for example, AT E Series boards), you must install the Windows NT 4.0 ISA Plug and Play driver. This driver is not installed by default. Follow these steps to install the driver:

1. Insert your Windows NT 4.0 CD.
2. Go the `\Drvlib\Pnpisa\X86` directory.
3. Right-click once on the `Pnpisa.inf` file, select the **Install** option, and follow instructions.
4. After you have installed the `Pnpisa.inf` file, restart your system.
5. Install your Plug and Play device in your computer.

To configure this device, use the NI-DAQ Configuration Utility to assign system resources. After you have made resource changes in the NI-DAQ Configuration Utility, you must restart your system.

PCI Devices

Install the PCI Plug and Play device in your computer. To configure this device, use the NI-DAQ Configuration Utility to assign system resources. After you have made resource changes in the NI-DAQ Configuration Utility, you must restart your system.

Configuring Plug and Play Devices in Windows 95

Install your Plug and Play device in your computer. The Windows 95 Device Manager allocates resources (base I/O address, interrupt, and DMA) to your Plug and Play device. Refer to the *NI-DAQ Configuration Utility Online Help* for detailed instructions.

After using the Windows 95 Device Manager to allocate resources, run the NI-DAQ Configuration Utility. This utility performs a full set of tests before saving the device configuration to ensure that the device operates correctly with the allocated resources. If the device fails any of the tests, the utility reports the errors and does not save the configuration. In this case, you need to start over by reinstalling the device in the **Device Manager** with different resources. After successfully testing and saving the device configuration in the NI-DAQ Configuration Utility, you do not need to restart your system.

In some cases, Windows 95 might classify a National Instruments Plug and Play DAQ device (for example, a PCI, PC Card, or ISA Plug and Play device) as a generic card and install it in the **Device Manager** under **Other Devices**. The device then will not be recognized by NI-DAQ as a valid DAQ device. This can occur if the card has been installed in the computer before you install NI-DAQ 5.1.

The following steps will correct the problem:

1. Install NI-DAQ 5.1 if you have not already done so.
2. Open the Windows 95 **Device Manager**. (From the **Start** menu, select **Settings»Control Panel**. From the **Control Panel**, double-click on **System**. In the **System Properties** dialog box, select the **Device Manager** tab.)
3. Under **Other Devices**, remove the entries that correspond to your type of Plug and Play DAQ device.

4. Restart your system. Windows 95 will re-identify all of your data acquisition devices. The devices will now appear in the Windows 95 under **Data Acquisition Devices**.

☞ **Note:** ***There might be more than one PCI card, PC Card, or ISA Plug and Play device classified under* Other Devices *in the Windows 95 Device Manager. If you are not sure which generic entry corresponds with your DAQ device (or devices), you might want to remove all the entries under* Other Devices, *restart, and let Windows 95 identify the devices again.***

Configuring Non-Plug and Play Devices on ISA (PC AT/XT) or EISA 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.

Consult your hardware user manual and check your device jumpers and DIP switches to verify that there are no resource conflicts, turn off your computer and plug in your devices. Then, continue with the SCXI installation instructions in your SCXI user manual, if applicable, and *Using The NI-DAQ Configuration Utility*.

☞ **Note:** ***You should not need to use the EISA configuration utility supplied with an EISA computer under any circumstances.***

Using The NI-DAQ Configuration Utility

The NI-DAQ Configuration Utility is a Windows-based application that you can use to configure and view National Instruments DAQ device settings.

☞ **Note:** ***To use the NI-DAQ Configuration Utility, you should quit any applications that are performing data acquisition operations.***

Refer to the *NI-DAQ Configuration Utility Online Help* for more information and detailed instructions. To run the help file, go to **Start»Programs»NI-DAQ for Windows»NI-DAQ Configuration Utility Help**.

Fundamentals of Building Windows Applications

Chapter 2

This chapter describes the fundamentals of creating NI-DAQ applications in Windows and Windows NT.

The following 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++
- 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 is stored in the executable files.

☞ **Note:** ***Use the 32-bit*** `NIDAQ32.DLL`***. If you are programming in C or C++, link in the*** `NIDAQ32.LIB` ***Import Library.***

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 can send information to the DLL routines through import libraries or through function declarations.

Using function prototypes is a good programming practice. That is why NI-DAQ is packaged with function prototype files for 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 three development tools that NI-DAQ works with, you must create your own function prototype file.

Creating a Windows Application Using Borland C++

This section assumes that you will be using the Borland IDE (Integrated Development Environment) 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 `.c` (C source code) or `.cpp` (C++ source code).
3. Set **Options\Application** to **Windows App** to set options similar to those used in a module definition file. You also can create an **EasyWinApp** if you are going to create a 16-bit application, or a **ConsoleApp** if you are recreating a 32-bit application. In these cases, you can skip the next step.
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 "NIDAQ.H"
   ```

3. Add the NI-DAQ import library `NIDAQBC.LIB` or `NIDAQ32B.LIB` to the project module, depending on the operating system you have.

Example Programs

You can find some example programs and project files in `.\Examples\BorlandC` in your NI-DAQ directory.

You can find additional information about these example programs in the *NI-DAQ Examples Online Help* file. You can access this file through the **Start»Programs»NI-DAQ for Windows»NI-DAQ Examples Online Help**.

Special Considerations

Buffer Allocation

To allocate memory, you can use the Windows functions `GlobalAlloc()` and `GlobalFree()`. After allocation, to use a buffer of memory, you must lock memory with `GlobalLock()`. After using the memory, you must unlock memory with `GlobalUnlock()`.

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

Huge Buffer Access for 16-Bit Programs

When referencing memory buffers that exceed 64 KB in size, use *huge* pointers to reference the buffer. Any other pointer type does not perform the correct pointer increment when crossing the 64 KB segment boundary.

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 C++ Integrated Development Environment (IDE) to manage your code development.

To develop NI-DAQ for with a graphical user interface, follow this procedure:

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

4. Create your resources. After you have created the resources, save them into an `.rc` (resource script) and add the file to the project.
5. If using Visual C++ 1.*x* (16-bit), remember to set the memory model to **Large**.
6. To use the NI-DAQ functions, prototype any NI-DAQ function used in your application by including the NI-DAQ header file in your source files, as shown below:

   ```
   #include "NIDAQ.H"
   ```

   You either can copy this file into your project directory, or you can specify the path to this header file in the `include` system environmental variable.
7. Add the NI-DAQ import library `NIDAQ.LIB` (for Visual C++ 1.*x*, 16-bit) or `NIDAQ32.LIB` (for Visual C++ 2.*x* or later, 32-bit) to the project module.
8. Build your application.

If you are getting started with NI-DAQ programming, you can create a Windows application without a graphical user interface; therefore, you will not need to create a module definition file or resource file. If you are using Visual C++ 1.*x* (16-bit), you can create **QuickWin** applications, and if you are using Visual C++ 2.*x* or later (32-bit), you can create *Console Applications*. These types of programs do not require the normal message and window handling as Window graphical user interface programs usually require, so your code is made much simpler.

To develop a *QuickWin* application using Visual C++ 1.*x* (16-bit), follow this procedure:

1. Create your source file of type `.c` (C source code) or `.cpp` (C++ source code), calling NI-DAQ functions as typical function calls. Remember to include the NI-DAQ header file, `NIDAQ.H`, as shown in the preceding list of instructions.
2. Open a new project file (**Project»New**) and specify the **Project Type** as **QuickWin application (.EXE)**. Remember to specify a project name and a directory.
3. Add the C or C++ source file you have created to the project. To do so, select the menu item **Project»Edit** and add your source files.
4. Add the 16-bit NI-DAQ import library `NIDAQ.LIB` to the project. Add this file in the same way you added your source files.

5. Set the **Memory Model** of the project to **Large**. To do so, select the menu item **Options»Project**, click on the compiler button, select **Memory Model**, and change the **Memory Model** to **Large**.
6. Build your application. To do so, select the menu item **Project»Build** (your project name)**.EXE**.

To develop a Console Application using Visual C++ 2.*x* or later (32-bit), follow this procedure (the specific steps depend on your version of Visual C++):

1. Create your source file of type `.c` (C source code) or `.cpp` (C++ source code), as you would for other Windows programs written in C or C++, calling NI-DAQ functions as typical function calls. Include the NI-DAQ header file, `NIDAQ.H` as shown in the preceding paragraphs.
2. Open a new project file (**File»New**) and specify the **Project Type** as **Console Application (.EXE)**. Specify a project name and a directory.
3. Add the C or C++ source file you have created to the project. To do this in Visual C++ 2.*x*, select the menu **Project»Files** and select the source files. In Visual C++ 4.*x*, select the menu item **Insert»Files Into Project** and select the source files.
4. Add the 32-bit NI-DAQ import library `NIDAQ32.LIB` to the project. Add this file in the same way you added your source files.
5. Build your application. To do this in Visual C++ 2.*x*, select the menu item **Project»Build** (your project name) **.EXE**. In Visual C++4.*x*, select the menu item **Build»Build** (your project name) **.EXE**.

Example Programs

You can find some example programs and project files in `.\Examples_Src\Visual C` in your NI-DAQ directory.

To load an example program, use one of the project modules with the `.MAK` extension.

If you are using Visual C++1.*x* (16-bit), go to the **Project** menu and select **Open**. Then select the `.MAK` project module of your choice.

If you are using Visual C++ 2.*x* or later (32-bit), go to the **File** menu, select **Open**, and select **List Files of Type** to be projects (`*.MAK`). Then select the `.MAK` project module of your choice. Refer to the

FILELIST.TXT file for a short description of all the files in the examples directory.

Special Considerations

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

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. If you are using Visual Basic 3.0, prototype any NI-DAQ routines used in your application. You can do this by adding the NI-DAQ header module NIDAQ_VB.BAS in the NI-DAQ.\include directory in your NI-DAQ directory.
   a. Go to the **File** menu and select the **Add File** option.
   b. Then, using the file dialog box, find NIDAQ_VB.BAS and click on the **OK** button.
   c. Verify the existence of the file in the *project* window. This header file prototypes all NI-DAQ functions.

☞ **Note:** ***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,*** *Introduction to NI-DAQ****, 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 some example programs and project files in the `.\Examples_src\Vbasic` directory in your NI-DAQ directory; the project files have a `.MAK` or `.VBP` extension.

If you are using Visual Basic 3.0, load the files by opening the **File** menu, selecting **Open Project**, and selecting the `.MAK` file of your choice. These are Visual Basic 2.0 projects, which you can open with Visual Basic 2.0 or later.

If you are using Visual Basic 4.0, you can load them opening the **File** menu, selecting **Open Project**, and selecting the `.VBP` file of your choice. These are Visual Basic 4.0 projects, which you can open only with Visual Basic 4.0 or later.

Refer to the `FILELIST.TXT` file for a short description of all the files in the examples directory.

Special Considerations

Buffer Allocation

Visual Basic 3.0 and 4.0 are both quite restrictive when allocating memory. You allocate memory by declaring an array of whatever data type with which you want to work. Visual Basic uses dynamic memory allocation so you can 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 (16-bit) integers, 16,384 (32-bit) integers, or 8,191 doubles).

To break the 64 KB buffer size barrier, you can use the Windows functions `GlobalAlloc()` and `GlobalFree()` to allocate and lock buffers larger than 64 KB. After allocation, to use a buffer of memory, you must lock memory with `GlobalLock()`. After using the memory, you must unlock memory with `GlobalUnlock()`.

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

The following paragraphs illustrate declarations of functions.

For Visual Basic 3.0 and 4.0 (16-bit):

```
Declare Function GlobalAlloc Lib “Kernel” (ByVal wFlags
As Integer, ByVal dwBytes As Long) As Integer
```

```
Declare Function GlobalFree Lib “Kernel” (ByVal hMem As
Integer) As Integer
```

```
Declare Function GlobalLock Lib “Kernel” (ByVal hMem As
Integer) As Long
```

```
Declare Function GlobalReAlloc Lib “Kernel” (ByVal hMem
As Integer, ByVal dwBytes As Long, ByVal wFlags As
Integer) As Integer
```

```
Declare Function GlobalUnlock Lib “Kernel” (ByVal hMem
As Integer) As Integer
```

For Visual Basic 4.0, 32-bit:

```
Declare Function GlobalAlloc Lib “kernel32” Alias
“GlobalAlloc” (ByVal wFlags As Long, ByVal dwBytes As
Long) As Long
```

```
Declare Function GlobalFree Lib “kernel32” Alias
“GlobalFree” (ByVal hMem As Long) As Long
```

```
Declare Function GlobalLock Lib “kernel32” Alias
“GlobalLock” (ByVal hMem As Long) As Long
```

```
Declare Function GlobalReAlloc Lib “kernel32” Alias
“GlobalReAlloc” (ByVal hMem As Long, ByVal dwBytes As
Long, ByVal wFlags As Long) As Long
```

```
Declare Function GlobalUnlock Lib “kernel32” Alias
“GlobalUnlock” (ByVal hMem As Long) As Long
```

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. Prepend the `ByVal` keyword if you need to pass by value.

Using Borland Delphi with NI-DAQ

The NI-DAQ installer installs a prototype file for use with Borland Delphi 2.0 or later, which is stored in the `.\INCLUDE` directory in your NI-DAQ directory. To use this prototype file, include the file `NIDAQ.PAS` into your Borland Delphi project, and be sure to include this line in your Delphi source code:

```
uses NIDAQ;
```

☞ **Note:** ***There are no examples for Borland Delphi. For examples on NI-DAQ function flow, refer to the examples of other languages to get started.***

NI-DAQ Examples

The NI-DAQ installer installs a suite of concisely written examples in the following languages:

- LabWindows/CVI
- Visual C++
- Visual Basic
- Borland C++

These examples illustrate how to use NI-DAQ functions to perform a single task. All examples are devoid of any code to extract values from graphical user interface (GUI) objects so that you can focus on how the code flow is formed. In addition, most parameters are hardcoded at the top of the routine so that if you decide to change them, you simply can change the assignment.

The examples correspond to the function flow charts that you will encounter in the following chapter. If a task and a flow chart in the following chapter suits your data acquisition needs, you should find a corresponding example to get you started.

Each example consists of the following files:

- an appropriate project file for the programming language (except for Borland C++)
- a single source code file to illustrate the task at hand
- a library of NI-DAQ example utility functions (for buffer creation, waveform plotting, error checking, and implementing a delay)

☞ **Note:** ***None of the examples are installed in their pre-compiled executable (*** `.EXE` ***) format. To run them, you first must build them or load them into the IDE for the appropriate programming language.***

The examples are stored in the hierarchy shown below for each language:

`.\AI` Analog Input examples

`.\AO` Analog Output examples

`.\DI` Digital Input examples

`.\DO` Digital Output examples

`.\CTR` Counter/timer examples

`.\SCXI` SCXI examples

The project files have the same filename (not including extension) as the source code files. The following types are installed:

- LabWindows/CVI:

  `.PRJ` (project file), `.C` (source file)

- Visual C++:

  `.MAK` (make file), `.C` (source file)

- Visual Basic:

  `.VBP` for VB 4.0 (32-bit) or `.MAK` for VB2 or later (16-bit), `.FRM` (form module)

- Borland C++:

  `.C` (source file)

For more information about each example, how to compile examples, and details on the NI-DAQ Example Utility functions, please refer to the *NI-DAQ Examples Online Help* file. To open this file, go to **Start»Programs»NI-DAQ for Windows»NI-DAQ Examples Online Help**.

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
- Event message
- Software-calibration and device-specific
- Analog input function group
  - One-shot analog input
    - Single-channel analog input
  - Data acquisition
    - High-level data acquisition
    - Low-level data acquisition
    - Low-level double-buffered data acquisition
- Analog output function group
  - One-shot analog output
  - Waveform generation
    - High-level waveform generation
    - Low-level waveform generation
- Digital I/O function group
  - Digital I/O
  - Group digital I/O
  - Double-buffered digital I/O
- Counter/Timer function group
  - Counter/timer
  - Interval counter/timer
  - General-purpose counter/timer

- RTSI bus trigger
- SCXI
- Transducer conversion

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 call it explicitly. 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.
`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`.

Software-Calibration and Device-Specific Functions

Each of these software-calibration and configuration functions is specific to only one type of device or class of devices.

`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, PCI-1200, Lab-PC-1200, Lab-PC-1200AI, 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.
`LPM16_Calibrate`	Calibrates the PC-LPM-16 and PC-LPM-16PnP 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 ADCs and DACs of MIO-F-5/16X devices. 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 MIO-F-5/16X devices.
`MIO_Config`	Turns dithering (the addition of Gaussian noise to the analog input signal) on and off. For the MIO-64, this function also lets you specify whether to use AMUX-64T channels or onboard channels.
`Select_Signal`	(E Series and DAQArb 5411devices only) Selects the source and polarity of certain signals used by the E Series and DAQArb 5411devices. 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.

Event Message Functions

NI-DAQ Event Message functions are an efficient way to monitor your background data acquisition processes, without dedicating your foreground process for status checking.

The NI-DAQ Event Message dispatcher notifies your application when a user-specified DAQ 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.

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.

Event notification can be done through user-defined callbacks and/or the Windows Message queue. When a user-specified event occurs, NI-DAQ calls the user-defined callback (if defined) and/or puts a message into the Windows Message queue, if you specified a window handle. 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.

☞ **Note:** ***You can use the VBXs in Visual Basic, versions 1.0, 2.0, 3.0 and 4.0 (16-bit) only. You can use the OCXs in Visual Basic, version 4.0 (32-bit) only.***

There are three NI-DAQ custom controls for Visual Basic applications:

- General Data Acquisition Event (`DAQEVENT.OCX`)

- Analog Trigger Event (`ATRIGEV.OCX`)

- Analog Alarm Event (`ALARMEV.OCX`)

All of these custom controls are placed in the `NIDAQ` subdirectory of your Windows 95/NT 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 if you use NI-DAQ custom controls shown previously in this section, 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 for PC Compatibles*
`DAQEvent`	0—Acquired or generated *n* scans 1—Every *n* scans 2—Completed operation or stopped by error 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`	N/A
`Tag`	N/A
`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 for PC Compatibles*.

You should set 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 `GeneralDAQEvent`*n*`.Refresh` method to set the DAQ Event in the NI-DAQ driver. Each subsequent invocation of `GeneralDAQEvent`*n*`.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. When the selected event occurs, the `GeneralDAQEvent`*n*`_Fire` procedure is 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 `GeneralDAQEvent`*n*`_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 for Windows* section.

Analog Trigger Event

You use the Analog Trigger Event custom control to configure and enable an analog trigger. See the *Event Message Functions* section 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 for PC Compatibles*
`Level`	Single (voltage)
`WindowSize`	Single (voltage)
`Slope`	0—Positive (default) 1—Negative
`TrigSkipCount`	Long
`PreTrigScans`	Long
`PostTrigScans`	Long
`Index`	N/A
`Tag`	N/A
`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 for PC Compatibles*.

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. When the Analog Trigger conditions are met, the `AnalogTriggerEvent`*n*`_Fire` procedure is 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* section 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)
`ChanStr`	See `Config_DAQ_Event_Message` in the *NI-DAQ Function Reference Manual for PC Compatibles*
`HighAlarmLevel`	Single (voltage)
`LowAlarmLevel`	Single (voltage)
`HighDeadbandWidth`	Single (voltage)
`LowDeadbandWidth`	Single (voltage)
`Index`	N/A
`Tag`	N/A
`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 for PC Compatibles*.

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 `AnalogAlarmEventn.Refresh` method to set the Analog Alarm Event in the NI-DAQ driver. Each subsequent invocation of `AnalogAlarmEventn.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 might 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:

- 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 then is distinguished by the Index property, and the event procedures is an extra parameter `Index as Integer`. The first element has `Index = 0`, the second element has `Index = 1`, and so on. You have only 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.

- You can place multiple controls from the **Visual Basic Tool Box** onto the form. Each individual custom control of the same type then is distinguished by the number after the `name` of the custom control, such as `GeneralDAQEvent1`, `GeneralDAQEvent2`, and so on. Consequently, you can 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. Error checking is not shown.

1. To use the GeneralDAQEvent custom control, you must first include the proper custom control file. If you are using Visual Basic, version 3.0 or earlier, select the **File»Add File** option, and look for `NI-EV100.VBX` in the `\Windows\SYSTEM` directory. If you are using Visual Basic 4.0 (32-bit), select the **Tools»Custom Controls** option, and select the National Instrument GeneralDAQEvent custom control.
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 see a small icon, which does not appear in run time.
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, use the `Set_DAQ_Device_Info` function to set the device analog inputs to use interrupts. 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 to the existing code:

```
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, 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 = "AI0"
GeneralDAQEvent1.Refresh                        'refresh to set params
er% = DAQ_Start(1, 0, 1, buffer%(0), 10000, 3, 10)
```

5. Next, define what to do when the DAQ Event occurs. In this example, we easily can update a text box upon every 1,000 scans, and also when the whole acquisition is done. Place a text box on your form. It automatically is 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. 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(_)`. 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 PC Compatibles* for guidance.
8. Click on the **Exit** button to stop the program.

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, single-channel analog input (`AI`) functions perform single A/D conversions on one channel. Within the data acquisition functions, there are four sets—high level, low level, and low-level double buffered.

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 the 516 devices, DAQCard-700, analog input Lab and 1200 devices, MIO and AI devices, and LPM devices:

`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 and MIO-F-5/16X devices, which have no jumpers for analog input configuration, this function programs the device for the settings you want. For the settings of 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 and MIO-F-5/16X devices, 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 you want. See Figure 3-1 for the function flow typical of single-point data acquisition. Also, refer to the *NI-DAQ Examples Online Help* (`NIDAQEX.HLP`) to find a related example.

When using SCXI as a front end for analog input to a DAQCard-700, analog input Lab and 1200 devices, MIO and AI device, or LPM devices, 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.

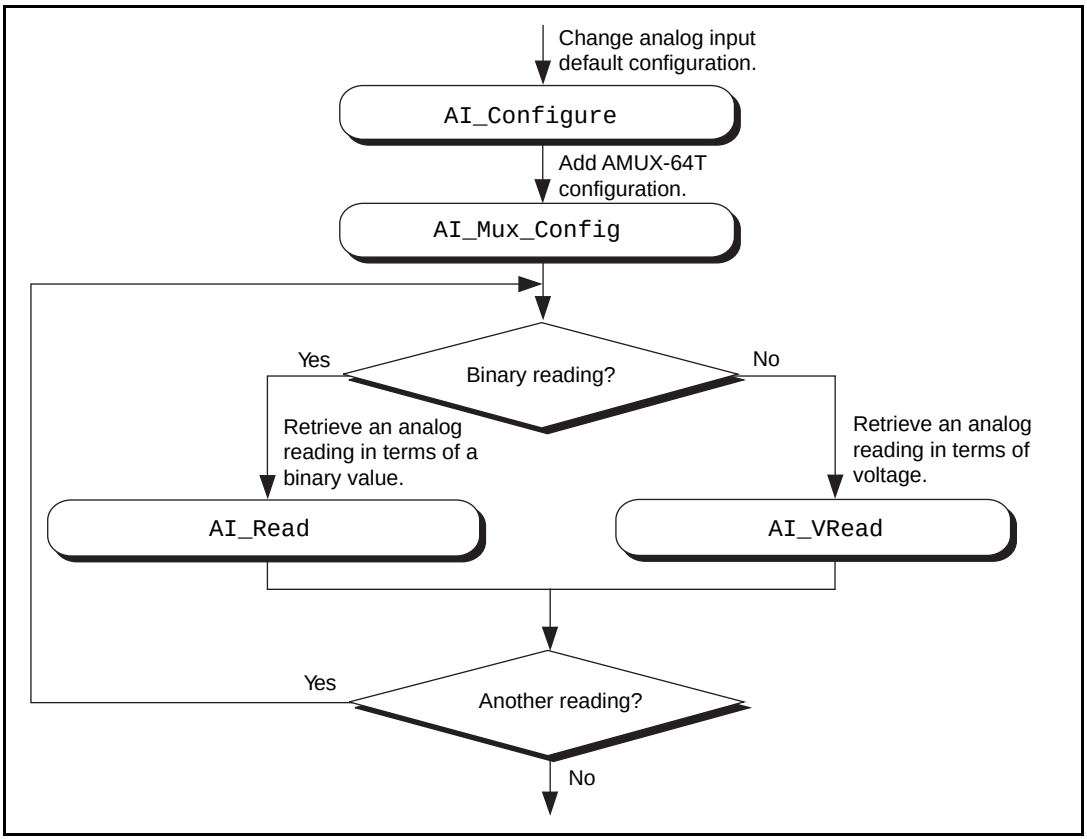


Figure 3-1. Single-Point Analog Reading with Onboard Conversion Timing

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 works only if your computer is faster than the rate of conversion pulses. Refer to the *Data Acquisition Functions* section later in this chapter to learn more about interrupt and DMA-driven data acquisition by using high-speed 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.

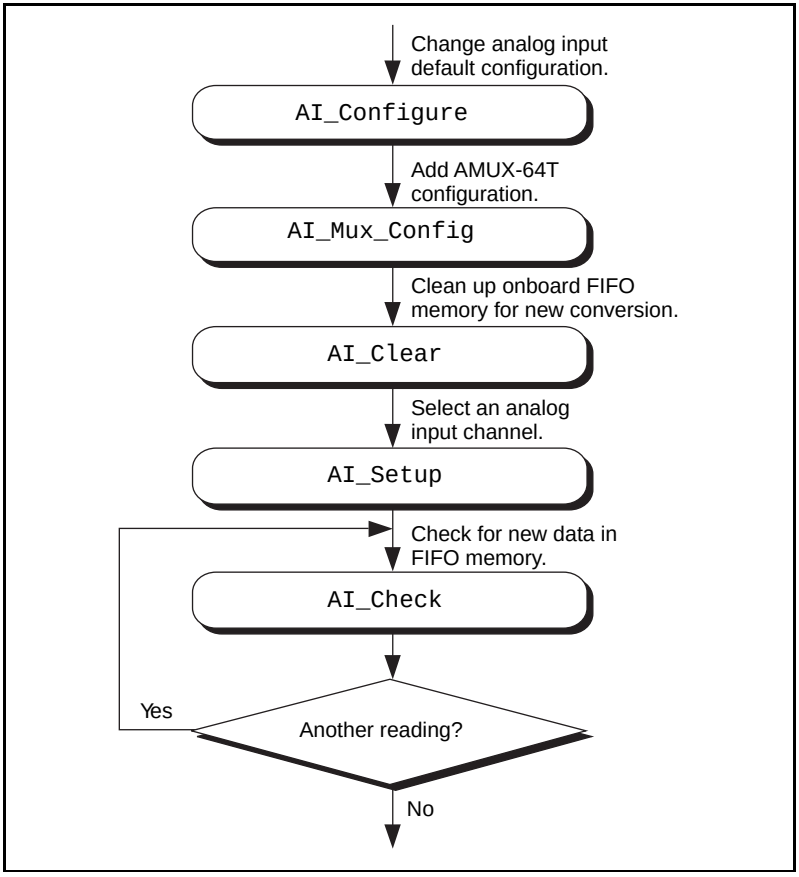


Figure 3-2. Single-Point Analog Reading with External Conversion Timing

Data Acquisition Functions

Use the `SCAN`, `DAQ`, `Lab`, and `DAQ_DB` functions with the following analog input devices:

- DAQCard-500/700
- 516 devices
- Lab and 1200 devices
- MIO and AI devices
- LPM devices

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 and saved all the data 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-500/700, 516 devices, LPM devices, and Lab and 1200 devices 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 all the data or an acquisition error has occurred (DAQCard-500/700, 516 devices, LPM devices, and Lab and 1200 devices 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).

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 rate you want.
`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 PF1 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-500/700, 516 devices, LPM devices, and Lab and 1200 devices only).
`Lab_ISCAN_Start`	Initiates a multiple-channel scanned data acquisition operation and stores its input in an array (DAQCard-500/700, 516 devices, LPM devices, and Lab and 1200 devices 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`	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 DAQ data acquisition operation, with or without interval scanning, and stores its input in an array (MIO and AI devices only).

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.
`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. This function 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

Lab and 1200 Devices Counter/Timer Signals

For the Lab and 1200 devices, 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.

DAQCard-500/700, 516 Devices, and LPM Device Counter/Timer Signals

For these devices, 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). Refer to Chapter 10, *AMUX-64T External Multiplexer Devices* in the *DAQ Hardware Overview Guide*, 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 DAQ data acquisition applications are made up of four building blocks, as shown in Figure 3-3. However, depending on the specific devices and applications you have, the NI-DAQ functions that make up each building block vary. Typical applications can include these NI-DAQ functions in each of their four building blocks.

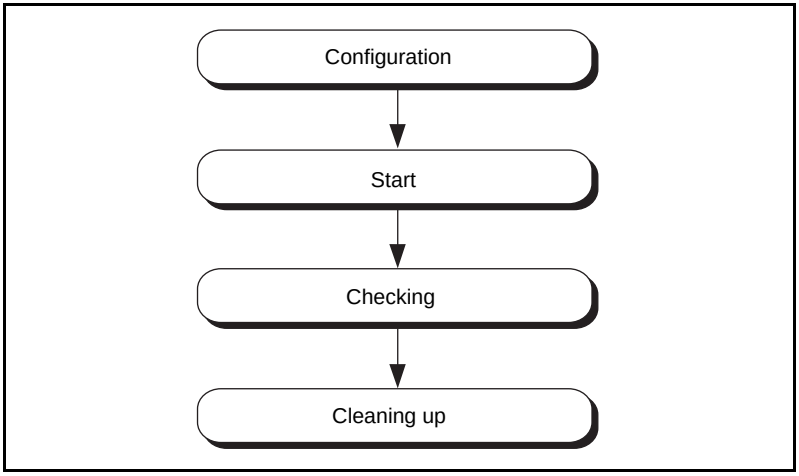


Figure 3-3. Buffered Data Acquisition Basic Building Blocks

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 3-4. However, you do not have to call all five functions every time you start a data acquisition.

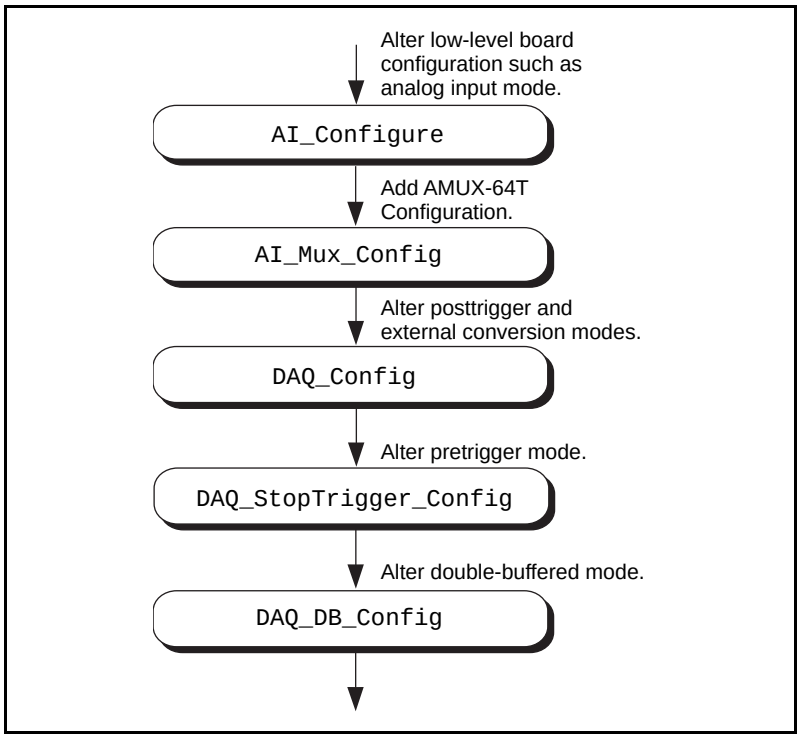


Figure 3-4. Buffered DAQ Data Acquisition Application Building Block 1, *Configuration*

NI-DAQ always records the device configurations and the default configurations. (See the `Init_DA_Brds` description in Chapter 2, *Function Reference*, of the *NI-DAQ Function Reference Manual for PC Compatibles*, 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-500/700, 516 devices, LPM devices, and Lab and 1200 devices only)
- `DAQ_to_Disk`
- `SCAN_to_Disk` (MIO and AI devices only)
- `Lab_ISCAN_to_Disk` (DAQCard-500/700, 516 devices, LPM devices, and Lab and 1200 devices 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 produce 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 might 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 calling program function caller immediately.

Asynchronous start functions include the `DAQ_Start`, `SCAN_Start`, and `SCAN_Start Lab_ISCAN_Start` functions. Figure 3-5 shows how the start calls make up building block 2.

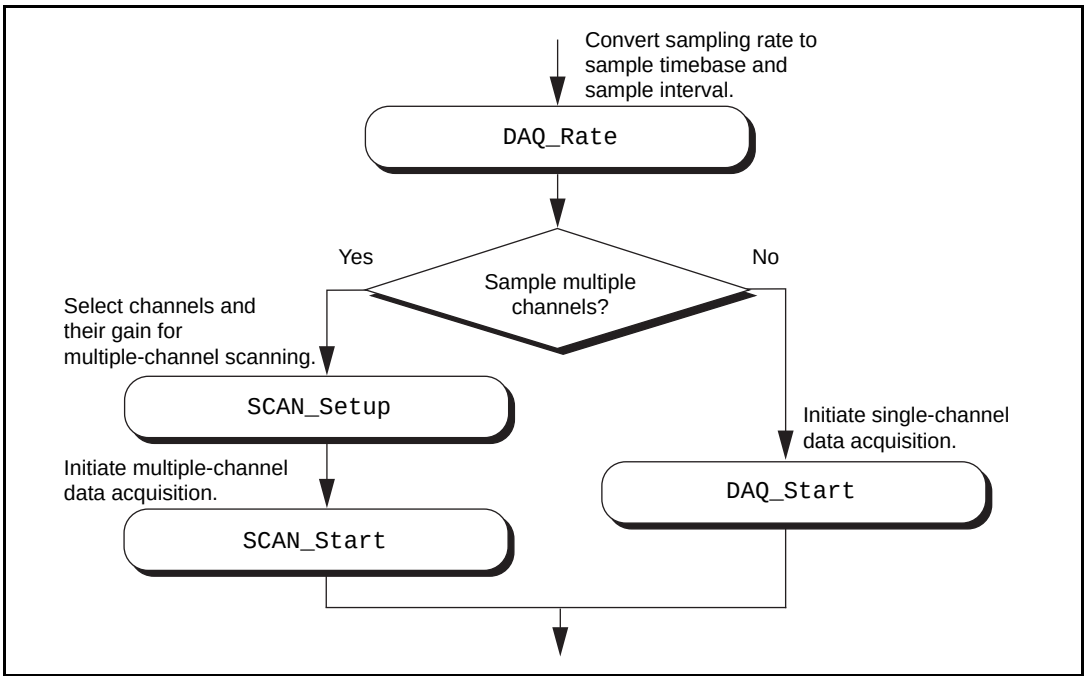


Figure 3-5. Buffered Data Acquisition Application Building Block 2, *Start*, for the MIO and AI Devices

If your device works with multirate scanning (scanning different channels at different rates), you can use `SCAN_Sequence_Setup` instead of `SCAN_Setup` in building block 2.

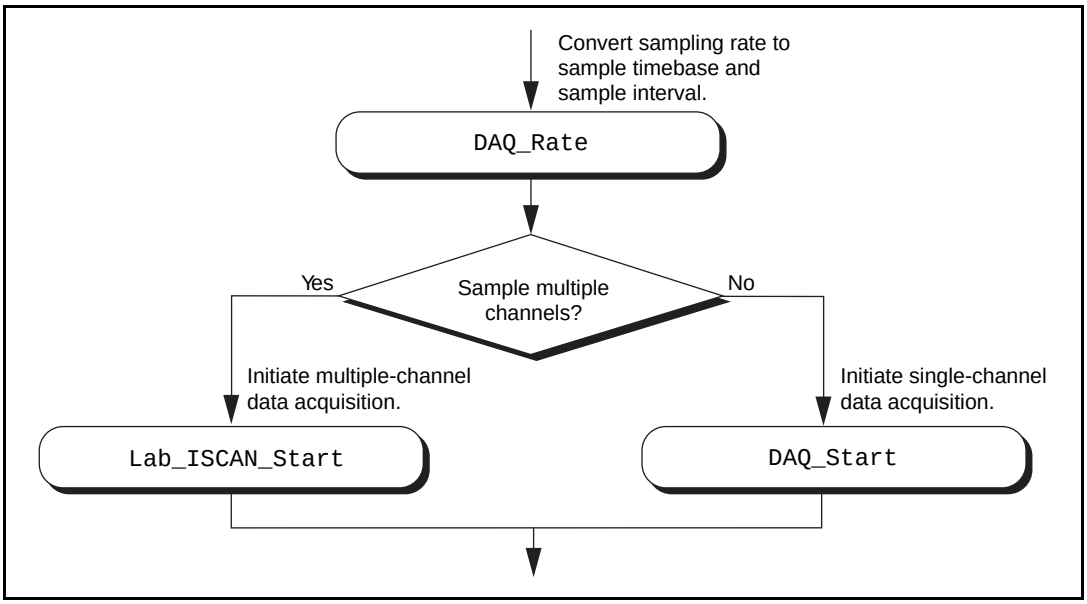


Figure 3-6. Buffered Data Acquisition Application Building Block 2, *Start*, for the 516 Devices, DAQCard-500/700, Lab and 1200 Devices, and LPM Devices

When you have the asynchronous start calls in your building block 2, the next building block, *Checking*, is very useful for finding out the status of the ongoing data acquisition process.

Building Block 3: Checking

`DAQ_Check` and `Lab_ISCAN_Check` shown in Figures 3-7 and 3-8 are simple and quick ways to check the ongoing data acquisition process. This call is often put in a while loop so that the application can monitor the data acquisition process periodically.

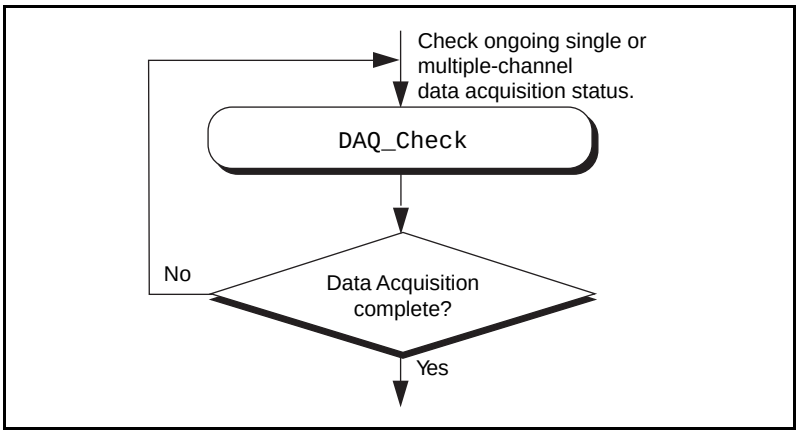


Figure 3-7. Buffered Data Acquisition Application Building Block 3, *Checking*, for the MIO and AI Devices

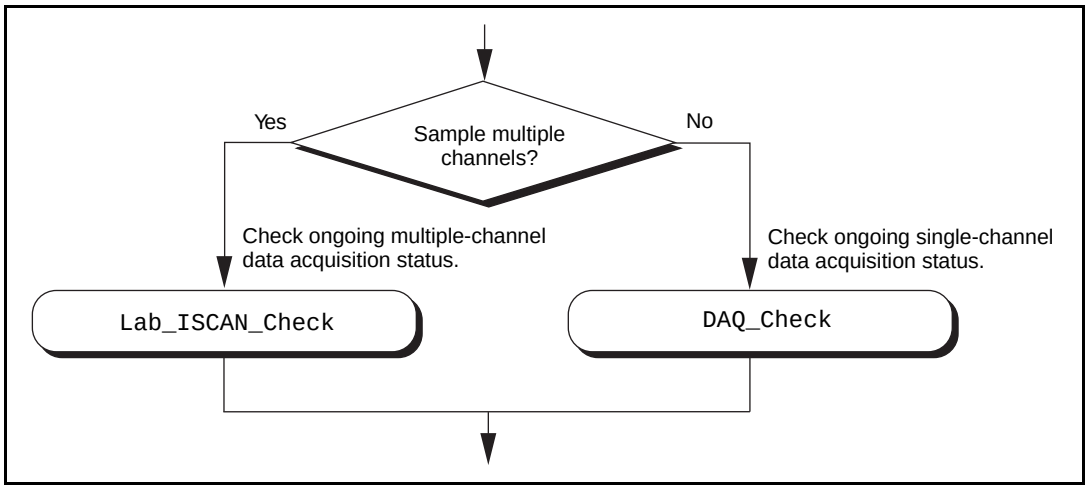


Figure 3-8. Buffered Data Acquisition Application Building Block 3, *Checking*, for the 516 Devices, DAQCard-500/700, Lab and 1200 Devices, and LPM Devices

However, if the information provided by `DAQ_Check` does not satisfy your needs, `DAQ_Monitor` or the double-buffered functions might be a better choice. With `DAQ_Monitor`, not only can you monitor the data acquisition process, but you also can 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 the double-buffered mode is enabled, all subsequent data acquisitions are in double-buffered mode.

In building block 3, different checking functions are needed. Figure 3-9 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 5 of this manual, *NI-DAQ Double Buffering*.

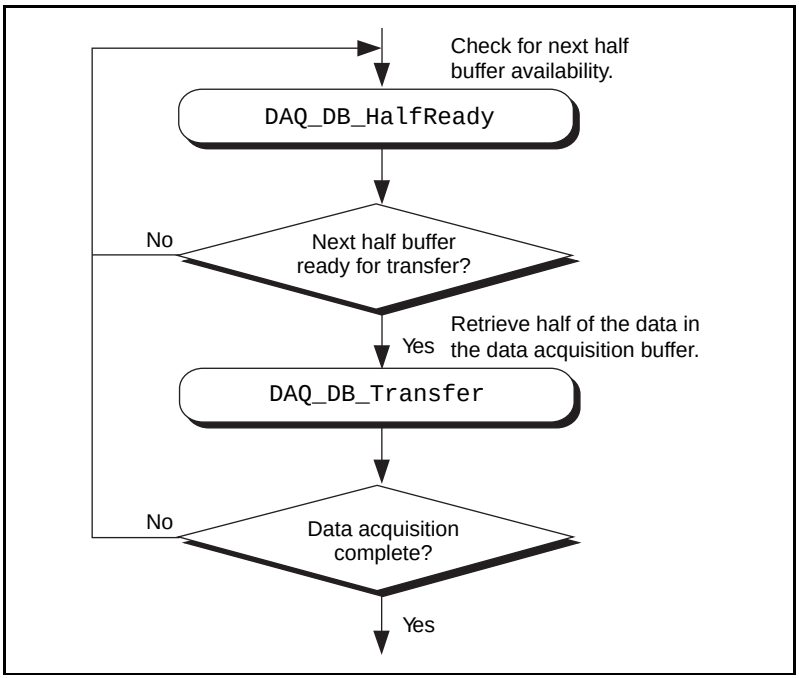


Figure 3-9. Double-Buffered DAQ Data Acquisition Application Building Block 3, *Checking*

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 MIO-F-5/16X devices 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`

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 3-10 shows how to use the multirate scanning functions in conjunction with other NI-DAQ functions.

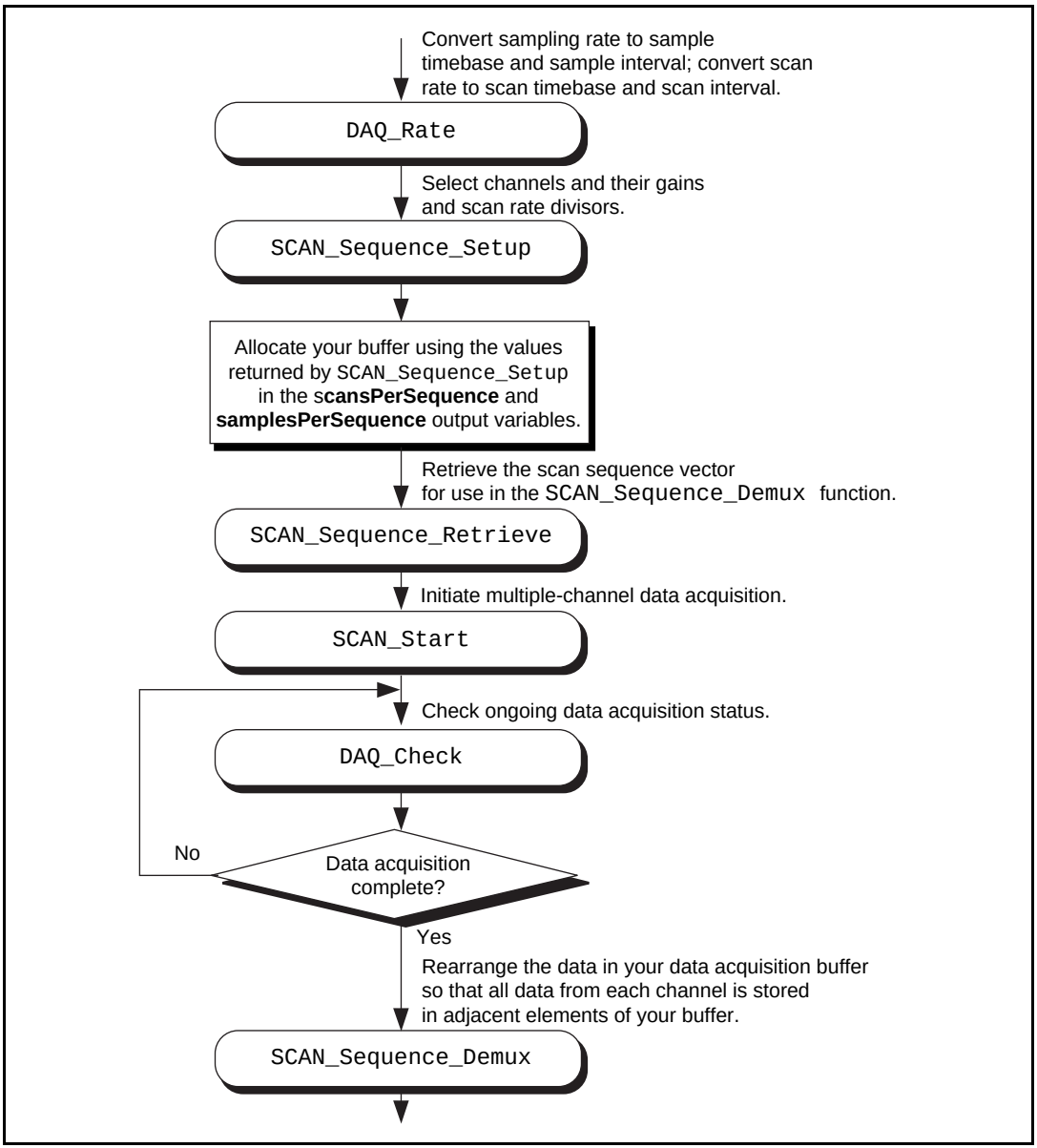


Figure 3-10. Multirate Scanning

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.

☞ **Note:** ***To use the SCXI-1124 analog output module, you must use the SCXI functions.***

One-Shot Analog Output Functions

Use the Analog Output functions to perform single D/A conversions with the MIO devices, AT-AO-6/10, VXI-AO-48XDC, AO-2DC devices, and the Lab and 1200 analog output devices:

`AO_Change_Parameter`	Selects a specific parameters setting for the analog output section or analog output channel. These parameters might be data transfer conditions, filter settings, or similar settings for a device.
`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 3-11 illustrates the equivalency.

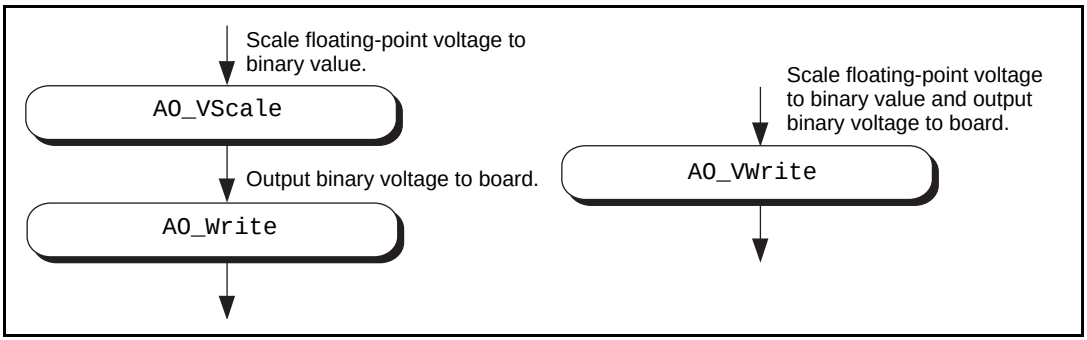


Figure 3-11. Equivalent Analog Output Calls

The following applications are shown using `AO_VWrite`. However, you can substitute the equivalent `AO_VScale` and `AO_Write` calls with no change in results.

Simple Analog Output Application

Figure 3-12 illustrates the basic series of calls for a simple analog output application.

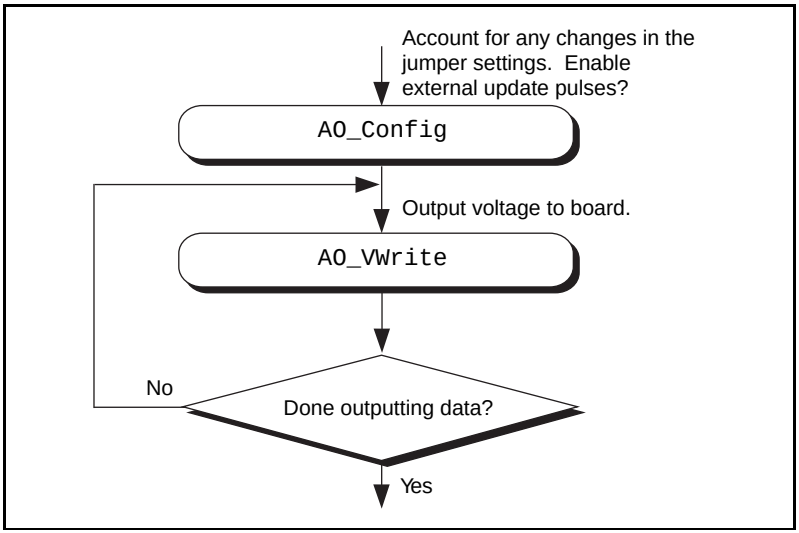


Figure 3-12. Simple Analog Output Application

The call to `AO_Configure` in Figure 3-12 has to be made only if you have changed the jumper settings of an MIO device, AT-AO-6/10, or Lab-PC+. You also might 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 an MIO-16/16D device, to pin 44 (EXTDACUPDATE*) on an MIO-F-5/16X device, to pin 48 (EXTUPDATE) on the AT-AO-6/10, to pin 39 (EXTUPDATE) on the Lab and 1200 analog output devices or to the selected pin on an MIO E Series device. You can change the voltages at all the analog output channels simultaneously.

The final steps in Figure 3-12 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 3-13 illustrates a modified version of the flowchart in Figure 3-12.

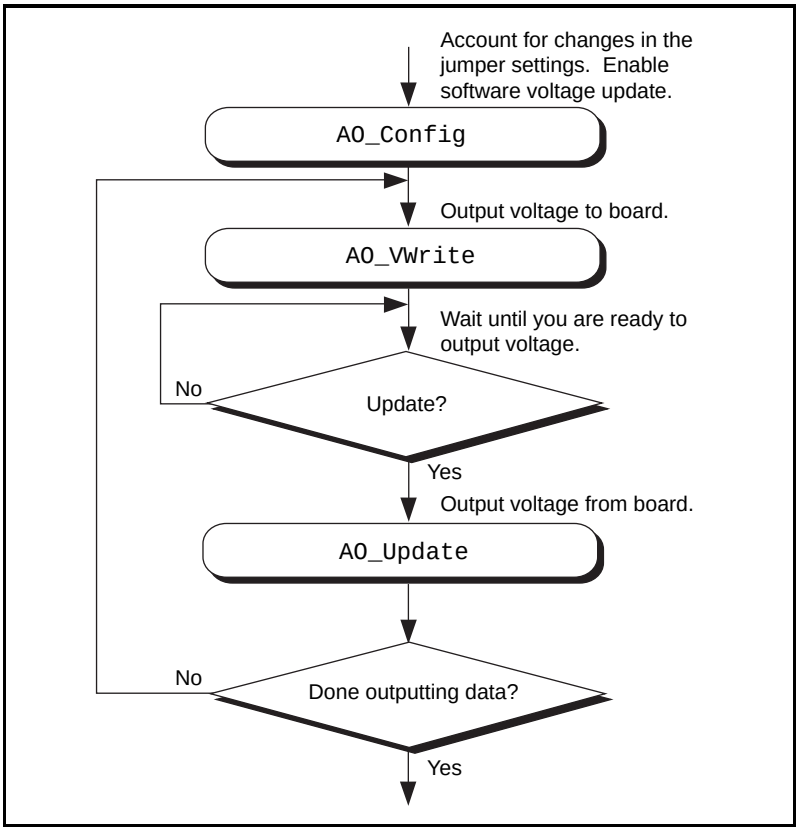


Figure 3-13. Analog Output with Software Updates

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 3-12.

☞ **Note:** ***Implement buffered analog output via the Waveform Generation functions.***

Waveform Generation Functions

Use the Waveform Generation functions to perform buffered analog output operations with the MIO devices, AT-AO-6/10 devices, and Lab and 1200 analog output devices.

High-Level Waveform Generation Functions

The following high-level Waveform Generation functions accomplish with a single call tasks that 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 the number of times the data in the file is to be generated, and starts the generation. `WFM_from_Disk` always waits for completion before returning, unless you call `Timeout_Config`.

`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`.

Low-Level Waveform Generation Functions

Low-level waveform generation functions are for setting up, starting, and controlling synchronous waveform generation operations:

`WFM_Chan_Control` Temporarily halts or restarts waveform generation for a single analog 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_Transfer`	Transfers new data into one or more waveform buffers (selected in `WFM_Load`) as waveform generation is in progress. `WFM_DB_Transfer` waits until NI-DAQ can transfer the data from the buffer to the waveform buffer.
`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 and 1200 analog output devices, and MIO devices 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. For DAQArb 5411devices, this function can enable the FIFO, ARB, or DDS

(Direct Digital Synthesis) mode of waveform generation and also can load the staging instructions to the device.

`WFM_Rate` Converts a waveform generation update rate into the timebase and update-interval values needed to produce the rate you want.

`WFM_Scale` Translates an array of floating-point values that represent voltages into an array of binary values that produce those voltages when NI-DAQ writes the binary array 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 outlines 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 3-14 illustrates the ordinary series of calls for a basic waveform application.

The first step of Figure 3-14 calls `WFM_Scale`. The `WFM_Scale` function converts floating-point voltages to integer values which produces the voltages (DAC values) you want.

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 rate you want.
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 PC Compatibles* 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 3-14 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.

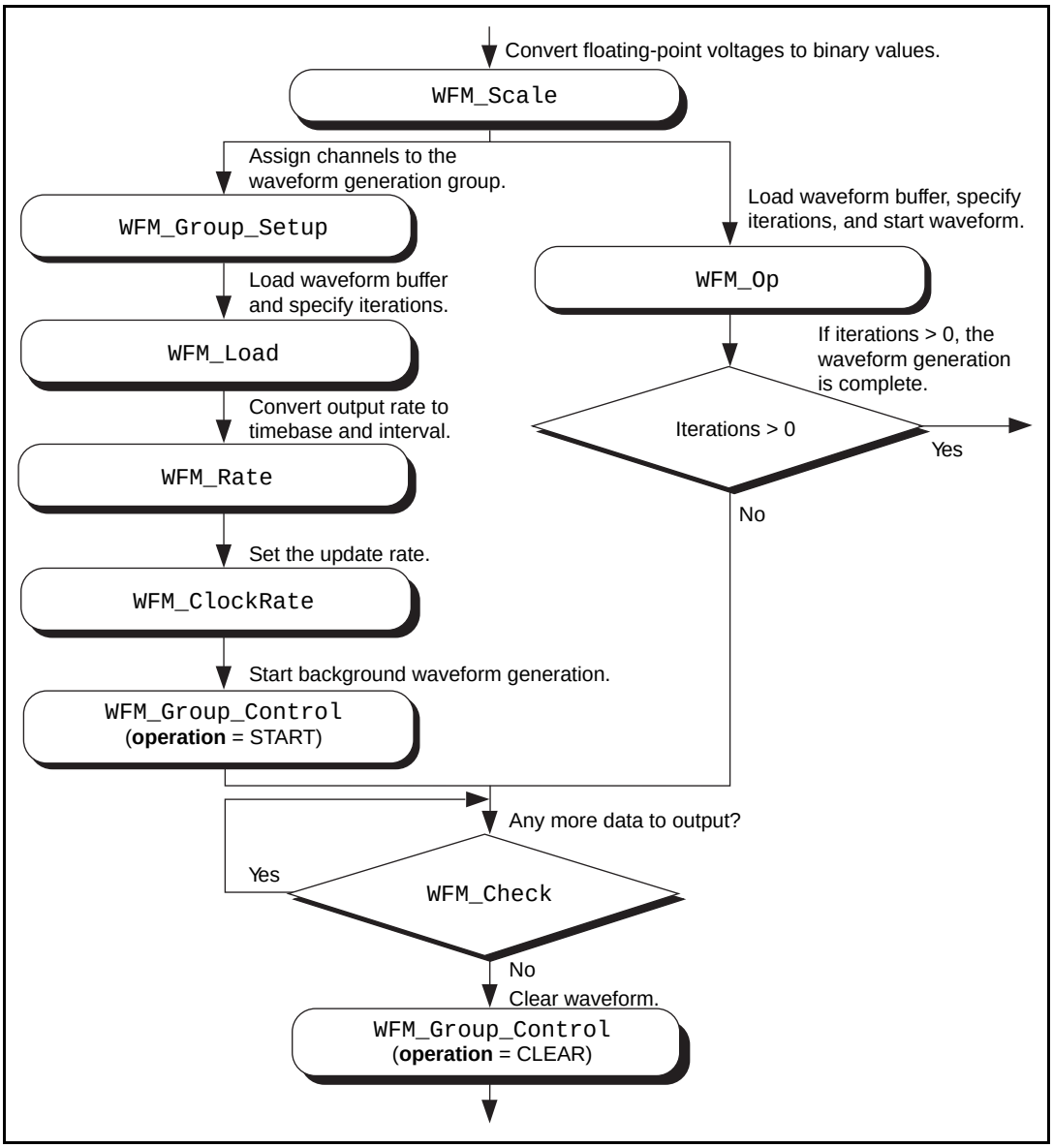


Figure 3-14. Basic Waveform Generation Application

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 3-15 illustrates the ordinary series of calls for a basic waveform application with pauses.

The first step of Figure 3-15 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 voltages you want. 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 rate you want. 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.

The next step is to assign an update rate to the group of channels using the `WFM_ClockRate` function. The `WFM_ClockRate` function assigns a timebase and update 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 PC Compatibles*, for further details.

Your application is now ready to start a waveform generation. Calling `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 3-15 is an application decision to pause the waveform generation. The application uses 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.

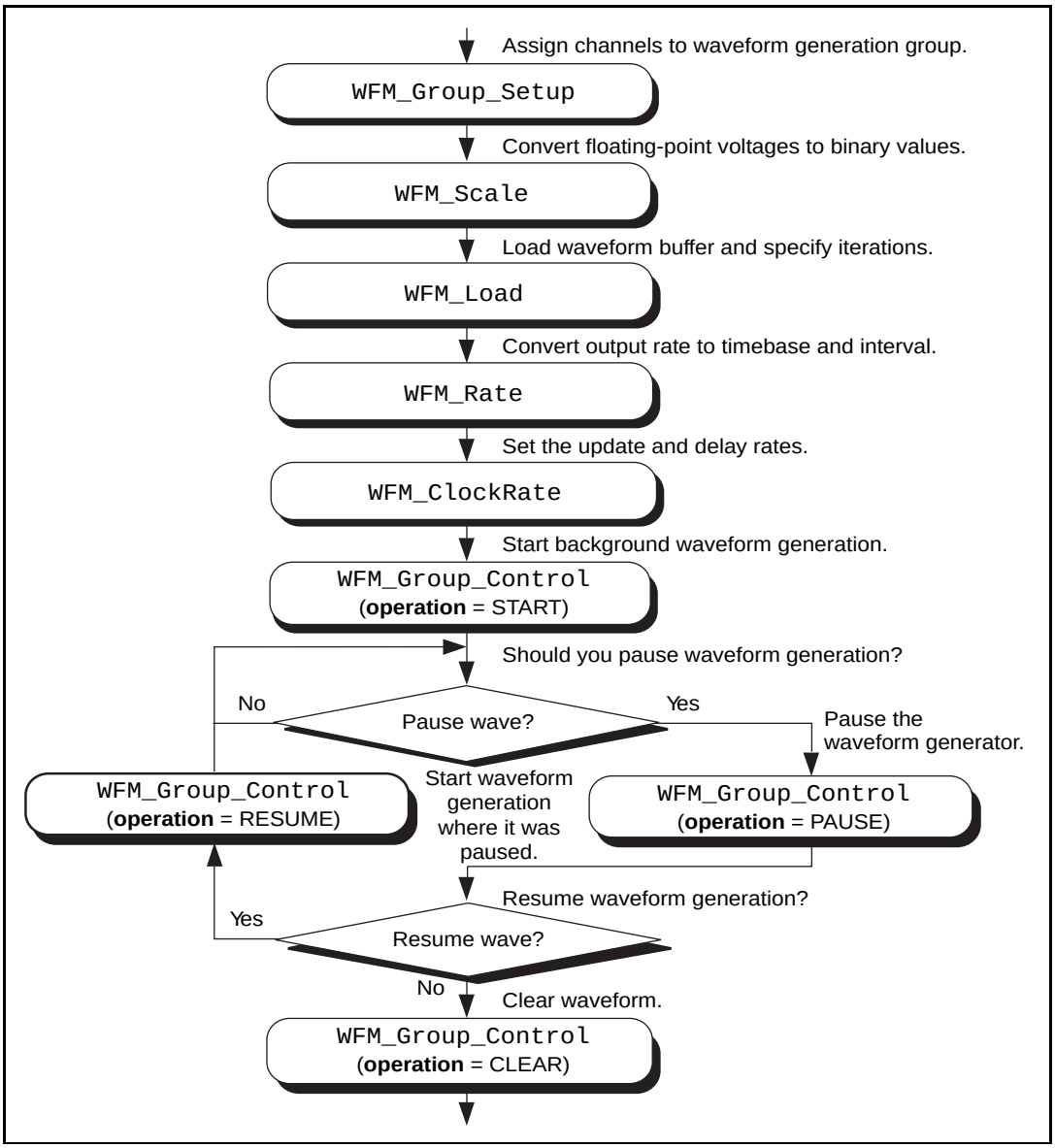


Figure 3-15. Waveform Generation with Pauses

Double-Buffered Waveform Generation Applications

You also can 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 3-16 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 3-16.

Although the step has 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. Calling `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 waits 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 can 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 halts any ongoing waveform generation.

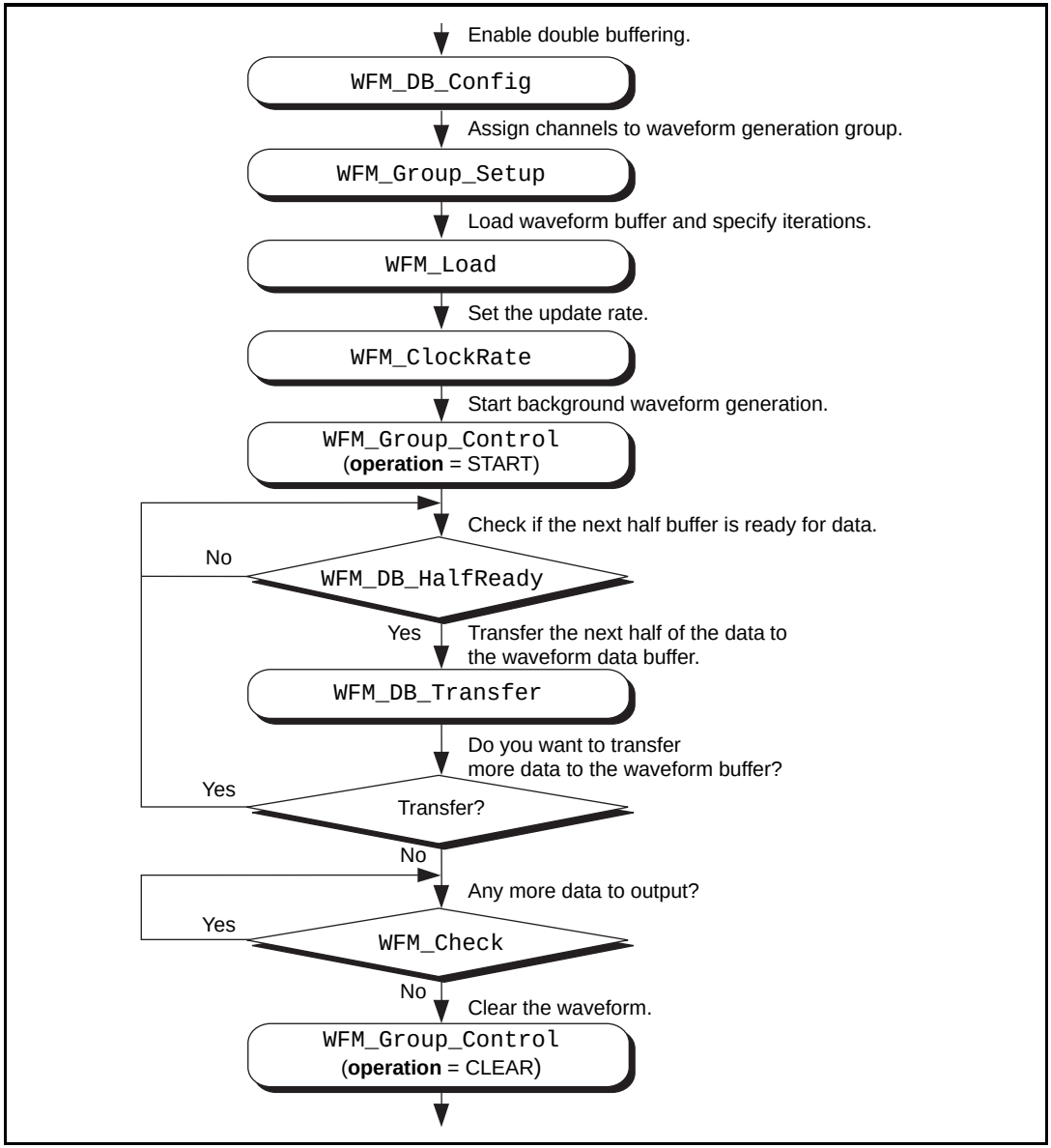


Figure 3-16. Double-Buffered Waveform Generation

Staging-Based Arbitrary Waveform Generation (For DAQArb 5411 Devices Only)

Figure 3-17 illustrates the series of calls for staging-based waveform generation. The *sequence list* is used in staging-based waveform generation for linking, looping, and generating multiple waveforms stored on the onboard memory. A sequence list contains a list of entries. Each entry is called a stage. Each stage specifies which waveform to generate and other associated settings for that waveform (for example, the number of loops in the generation).

The first step in Figure 3-17 to call the `WFM_Group_Setup` function. This function assigns one or more analog output channels to a group.

The second step is to set up trigger modes, analog filter, digital filter, output enable, attenuation, output impedance, SYNC duty cycle, PLL reference frequency, and other miscellaneous settings that the `AO_Change_Parameter` call. Refer to the `AO_Change_Parameter` function description in Chapter 2, *Function Reference*, of the *NI-DAQ Function Reference Manual for PC Compatibles* for more details.

The third step is to assign a waveform buffer to the analog output channels using the `WFM_Load` function. Refer to the `WFM_Load` function description in Chapter 2, *Function Reference*, of the *NI-DAQ Function Reference Manual for PC Compatibles* for more details.You can use two types of waveform buffer modes for staging. When **mode**=2, the buffer is a DDS (Direct Digital Synthesis) buffer, and only one buffer is loaded onto the board. When **mode**=3, the buffer is an ARB buffer, and you can load multiple buffers onto the board and assign them an ID.

You can call `WFM_Load` a number of times to load different buffers consecutively. When you call this function, you must assign each buffer an ID using the **iterations** parameter. The first buffer should have an ID of 0, and the successive buffers always should have the buffer ID of one more than the previously loaded buffer. When all of the buffers are loaded, you need to load the sequence list. To generate a sequence of these buffers in the order you want, call `WFM_Load` again with **mode**=4.

The fourth step is to assign an update rate to the group of channels by calling the `WFM_ClockRate` function. This function assigns a timebase and update interval to a group of analog output channels. You can learn more about the `WFM_ClockRate` function in Chapter 2, *Function Reference*, of the *NI-DAQ Function Reference Manual for PC Compatibles*. Your application is now ready to start a waveform generation.

You use the `WFM_Group_Control` (**operation**=START) to start the waveform generation. The waveform generation continues, depending on the trigger mode what you set up earlier.

The final step is to call `WFM_Group_Control` (**operation**=CLEAR). The CLEAR operation halts any ongoing waveform generation. This operation also performs all of the necessary clean-up work after a waveform generation.

☞ **Note:** ***If you want to load another sequence list for waveform generation, which uses some or all of the buffers already loaded onto the board, you do not have to load the waveform buffers again. Instead, you can call*** `WFM_Load` ***again with*** **mode=4** ***with the new sequence list.***

If you want to load any new waveform buffer onto the board, you should start from the beginning by calling `WFM_Load` with **mode**=2 or **mode**=3 and the **iterations** parameter set to 0. All information about the buffers loaded previously will be cleared.

☞ **Note:** ***You can change the settings of the analog filter, digital filter output enable, attenuation, output impedance, SYNC duty cycle, or filter correction frequency even when the waveform generation is in progress.***

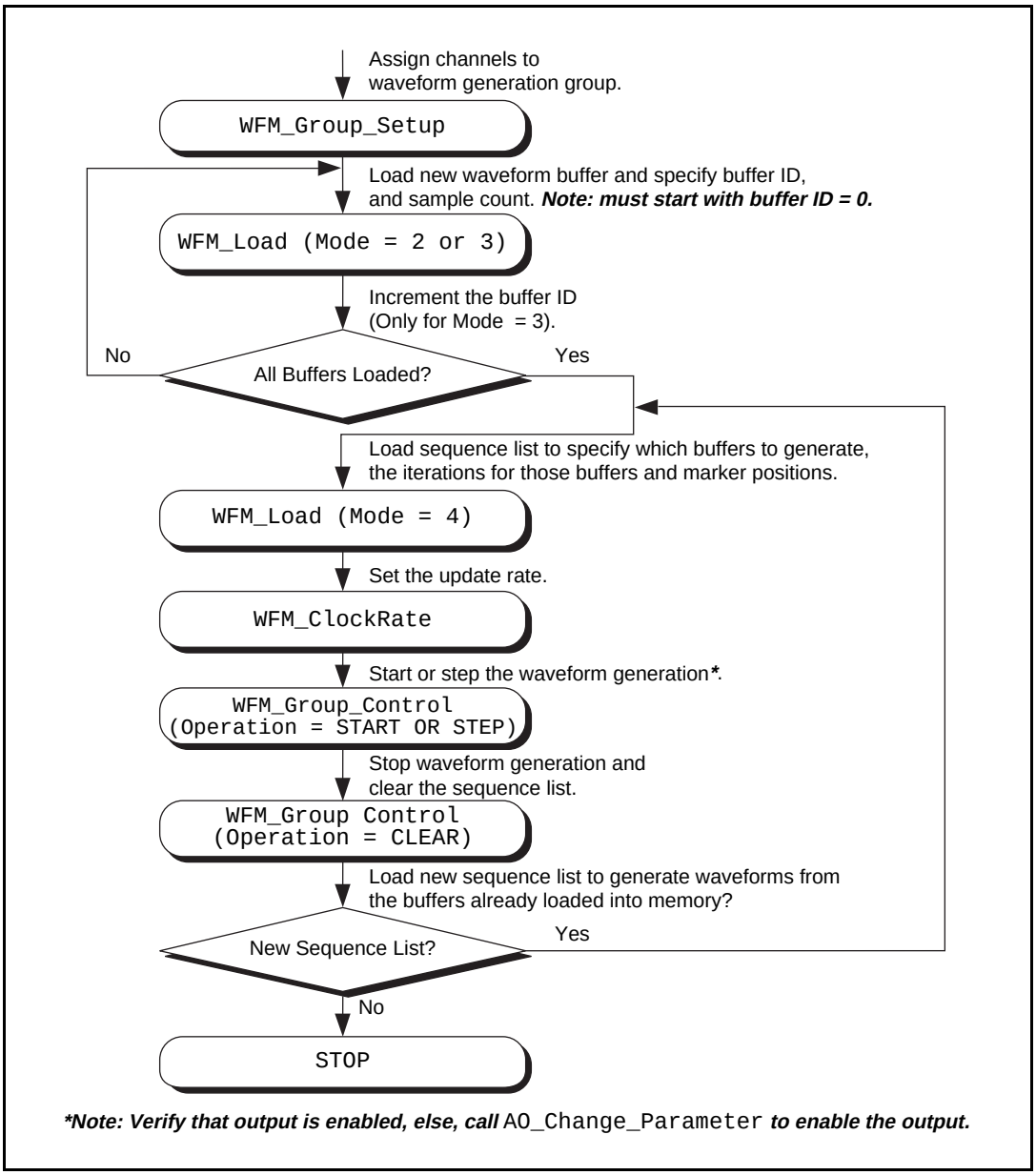


Figure 3-17. Staging-Based Waveform Generation

Reference Voltages for Analog Output Devices

Table 3-4 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**	**65,535**
AT-MIO-16X, AT-MIO-16XE-10, PCI-MIO-16XE-10, PCI-MIO-16XE-50, PCI-6031E (MIO-64XE-10), VXI-MIO-64XE-10	0 V	—	Reference voltage
All other MIO devices	0 V	Reference voltage	—
AT-AO-6/10	0 V	Reference voltage (+10 V in default case)	—
Lab and 1200 devices with analog output	0 V	+5 V	—

Table 3-5 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, AT-MIO-16XE-10, PCI-MIO-16XE-10, PCI-MIO-16XE-50, PCI-6031E (MIO-64XE-10), VXI-MIO-64XE-10	—	—	Negative of the reference voltage	Reference voltage
All other MIO devices	Negative of the reference voltage	Reference voltage	—	—

Table 3-5. Output Voltages with Bipolar Output Polarity (Continued)

Device	Value in Waveform Buffer			
	-2,048	**2,047**	**-32,768**	**32,767**
AT-AO-6/10	Negative of the reference voltage (-10 V in default case)	Reference voltage (+10 V in default case)	—	—
Lab and 1200 devices with analog output	-5 V	+5 V	—	—
DAQArb AT-5411 DAQArb PCI-5411	—	—	-5 V into 50 Ω or -10 V into an unterminated load	5 V into 50 Ω or -10 V into an unterminated load

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 affect DMA waveform generation of the AT-MIO-16F-5 adversely. 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 depends 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 allow `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 PC Compatibles* for more information.

The first point of the waveform buffer is not generated until the second update pulse occurs. That is, two update intervals 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 either should 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 and E Series devices, dedicated counters from the DAQ-STC chip are used for control and timing of waveform generation.

For the MIO-16/16D devices, 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 MIO-F-5/16X devices, 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 and 1200 devices and analog output devices, 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 produces 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 produce 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 produces 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 Series, 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 because of 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.

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 analog output FIFO size.

With PCI E Series devices in NI-DAQ 5.1 and later, you can reduce or even eliminate the FIFO lag effect by specifying the FIFO condition NI-DAQ uses to determine when to put more data into the FIFO. Refer to the `AO_Change_Parameter` function in the *NI-DAQ Function Reference Manual for PC Compatibles* for details.

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 and E Series devices, see the `Select_Signal` function description in the *NI-DAQ Function Reference Manual for PC Compatibles*.

On Am9513-based MIO 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*.

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, including all 65*33* devices
- All MIO and AI devices
- AT-AO-6/10
- 516 devices, DAQCard-500/700, LPM devices
- Lab and 1200 devices
- PC-TIO-10
- AO-2DC devices
- VXI-AO-48XDC

To use the SCXI digital and relay modules, you should 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 combine digital I/O ports into a larger entity called a *group*. On the DIO-32F and DAQDIO 6533 (DIO-32HS) devices, 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:

Table 3-6. Mapping of a Byte to Digital I/O Lines

Bit Number	Digital I/O Line Number
7	7 Most significant bit (MSB)
6	6
5	5
4	4
3	3
2	2
1	1
0	0 Least significant bit (LSB)

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, DAQDIO 6533 (DIO-32HS) devices, and on E Series devices (except for ports 2, 3, and 4 on the AT-MIO-16DE-10), you can program lines on the same port independently 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 (ports 2 and 3 only), DIO-96, Lab and 1200 devices, DAQDIO 6533 (DIO-32HS) devices and DIO-32F for handshaking and no-handshaking modes. These two modes have the following characteristics:

- No-handshaking 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 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.

☞ **Note:** ***On the DAQDIO 6533 (DIO-32HS) and DIO-32F devices, you must assign ports to a group before you can use handshaking mode.***

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, DIO-96, and Lab and 1200 Devices Groups

You can group together any combination of ports 0, 1, 3, 4, 6, 7, 9, and 10 on the DIO-96, of ports 0 and 1 on the DIO-24 and Lab and 1200 devices, 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 PC Compatibles* for more details.

DIO-32F and DAQDIO 6533 (DIO-32HS) Groups

On the DIO-32F and DAQDIO 6533 (DIO-32HS) devices, you can assign ports 0 through 3 (referring to ports DIOA through DIOD) 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 your device user manual for more information on the handshaking signals.

A group can be 8, 16, or 32 bits wide. An 8-bit group can be port 0, 1, 2, or 3. A 16-bit group can be ports 0 and 1 or ports 2 and 3. A 32-bit group is all four ports.

After you have assigned ports to a group, the group acts as a single entity controlling 8, 16, or 32 digital lines simultaneously. The DIO-32F has certain restrictions on which ports can be assigned to which groups. Refer to Table 3-7 for details.

Table 3-7. Legal Group Assignments for DIO-32F Devices

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 a handshaking protocol, 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 for PC Compatibles****.***

Digital I/O Functions

The Digital I/O (`DIG`) functions perform nonhandshaked digital line and port I/O for the AO-2DC, DIO, DAQDIO 6533, MIO, AI, 516, Lab and 1200 devices, LPM devices, AT-AO-6/10, DAQCard-500/700, VXI-AO-48XDC, and PC-TIO-10. The Digital I/O functions also perform handshaked port I/O for the DIO-24, AT-MIO-16D, AT-MIO-16DE-10, Lab and 1200 devices, and DIO-96:

`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 DAQDIO 6533 (DIO-32HS), PC-TIO-10, VXI-AO-48XDC, 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, DIO-96, Lab and 1200 devices only).

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, DIO-96, Lab and 1200 devices, AT-MIO-16D, AT-MIO-16DE-10, DAQDIO 6533 (DIO-32HS) devices, and AT-DIO-32F:

`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 and DAQDIO 6533 devices only).
`DIG_Grp_Config`	Configures the specified group for port assignment, direction (input or output), and size (DIO-32F and DAQDIO 6533 devices only).
`DIG_Grp_Mode`	Configures the specified group for handshake signal modes (DIO-32F and DAQDIO 6533 devices only).
`DIG_Grp_Status`	Returns a status word indicating the handshake status of the specified group (DIO-32F and DAQDIO 6533 devices only).
`DIG_In_Grp`	Reads digital input data from the specified digital group (DIO-32F and DAQDIO 6533 devices only).
`DIG_Out_Grp`	Writes digital output data to the specified digital group (DIO-32F and DAQDIO 6533 devices 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, DIO-96, and Lab and 1200 devices only).
`DIG_Trigger_Config`	Enables or disables the trigger mode of buffered digital I/O to indicate when to start and stop the data acquisition (DAQDIO 6533 devices only).

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 `DIG_DB_Transfer` waits until the data can be transferred before returning.
`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 outlines 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.

Handshaking Versus No-Handshaking 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 no-handshaking mode. The second method is to coordinate digital data transfers with another digital port. The second method is called 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 handshaking and no-handshaking 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 configured for handshaking or no-handshaking. All AT and PC devices having digital I/O ports can use no-handshaking digital port I/O. DIO-24, AT-MIO-16D, AT-MIO-16DE-10, DIO-96, and Lab and 1200 devices also can execute handshaking digital I/O for using the port I/O functions.

Figure 3-18 illustrates the series of calls for digital port I/O applications without handshaking. Figure 3-19 illustrates the series of calls for digital port I/O applications with handshaking.

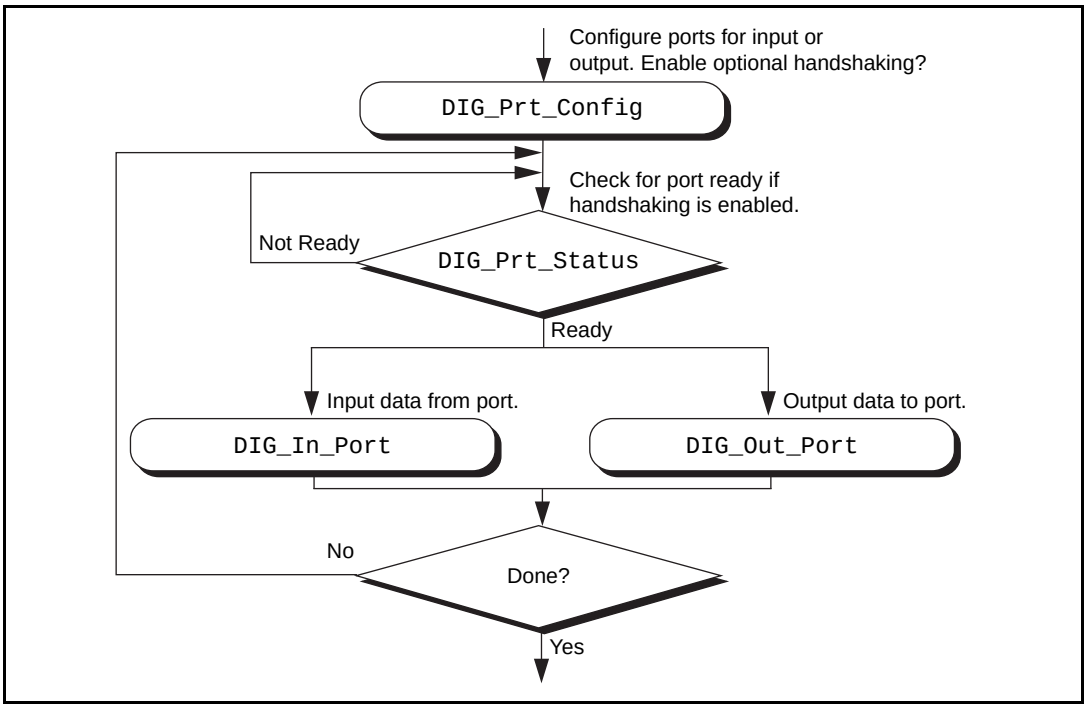


Figure 3-18. Basic Port Input or Output Application with Handshaking

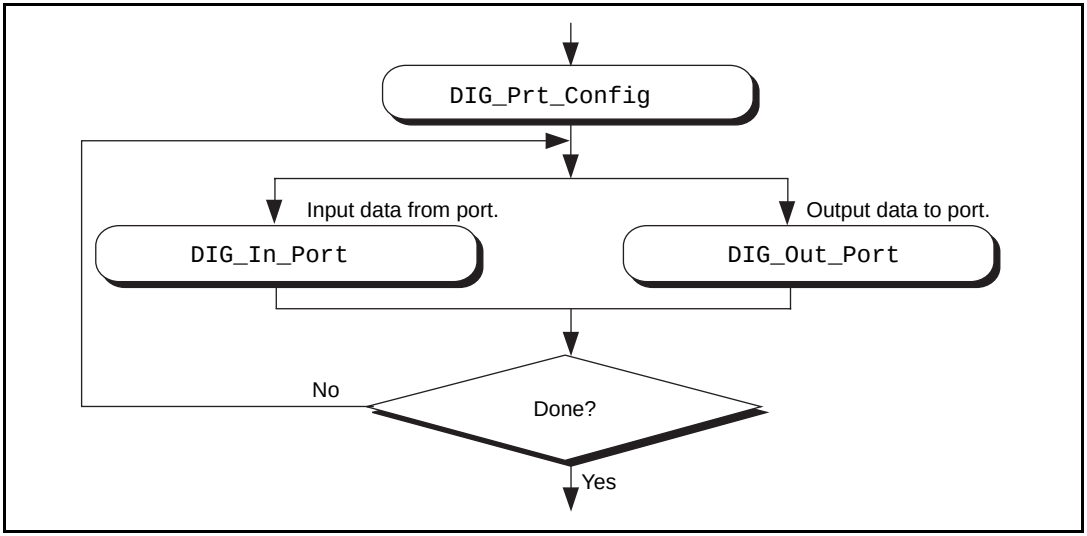


Figure 3-19. Basic Port Input or Output Application without Handshaking

The first step is to call `DIG_Prt_Config`, with which you configure the 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 3-19). If handshaking is enabled, 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 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 an output 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 only can transfer data in no-handshaking mode.

Figure 3-20 is a flowchart outlining the basic line I/O application.

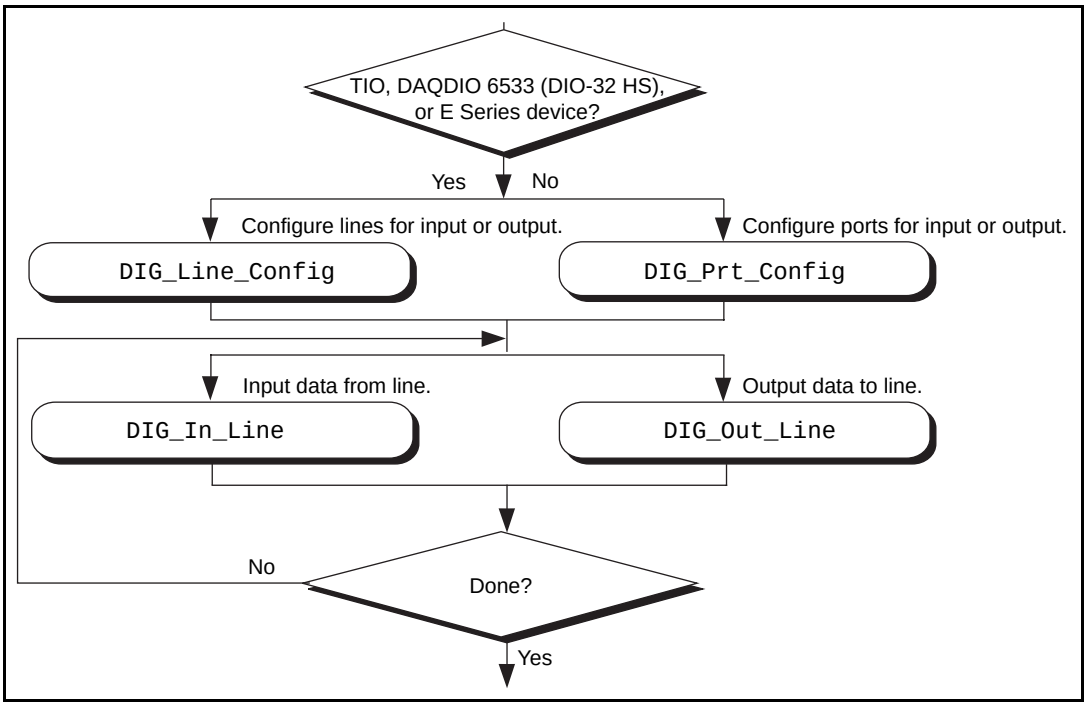


Figure 3-20. Basic Line Input or Output Application

The first step is to configure the digital lines for input or output. You can program DAQDIO 6533 (DIO-32HS) devices, 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 3-21 is a flowchart for group digital applications that input and output data one point at a time. Only the DIO-32F and DAQDIO 6533 (DIO-32HS) devices can execute group input or output one point at a time.

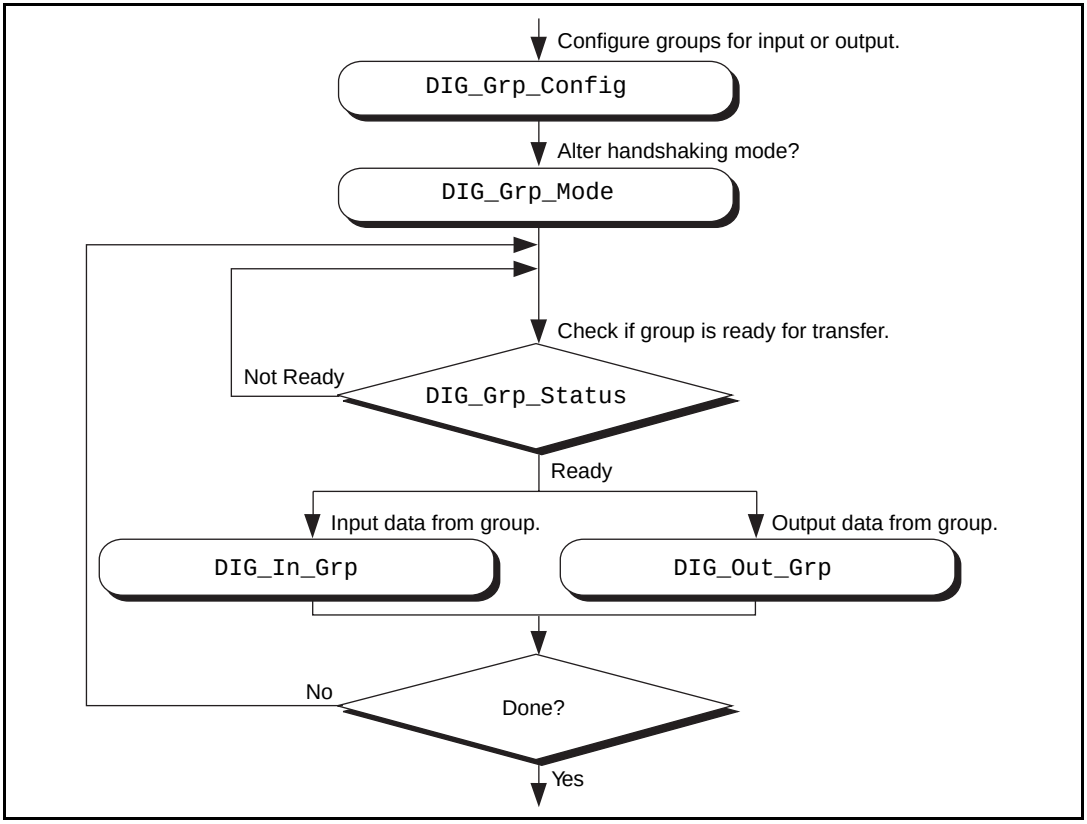


Figure 3-21. Simple Digital Group Input or Output Application

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-32F and DAQDIO 6533 devices by calling `DIG_Grp_Mode` (step 2 of Figure 3-21). 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 3-21). 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 3-22 outlines the basic steps for applications that use block I/O.

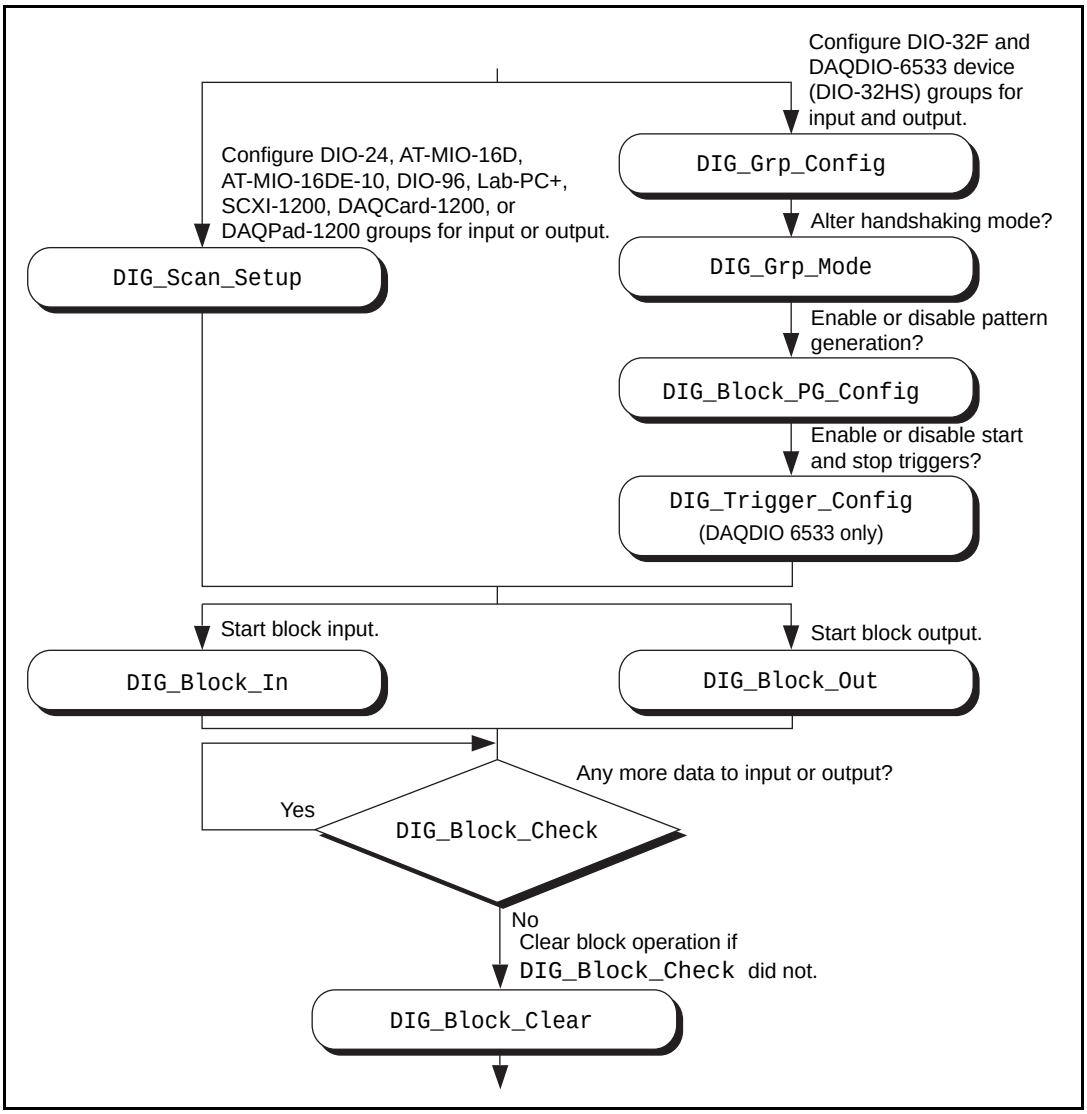


Figure 3-22. Digital Block Input or Output Application

☞ **Note:** ***The DIO-32F, DAQDIO 6533 (DIO-32HS), DIO-24, AT-MIO-16D, AT-MIO-16DE-10, DIO-96, and Lab and 1200 devices all can perform group block operations. However, the DIO-24, AT-MIO-16D, AT-MIO-16DE-10, DIO-96, and Lab and 1200 devices 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 and DAQDIO 6533 (DIO-32HS) devices.***

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. Call `DIG_Grp_Config` if you have a DIO-32F or a DAQDIO 6533 device. Call `DIG_SCAN_Setup` for all other devices. The DIO-32F is restricted to group sizes of two and four ports for block I/O.

If you are using a DIO-32F or DAQDIO 6533 device, you either can alter the handshaking mode of the group by calling `DIG_Grp_Mode` or perform digital pattern generation by calling `DIG_Block_PG_Config`, as shown in Figure 3-22. 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 3-22. 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 3-20, is to call `DIG_Block_In` or `DIG_Block_Out` to 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` automatically calls `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 also can configure group block operations as double-buffered operations for DIO-32 devices. With double-buffered operations, you can do continuous input or output with a limited amount of memory.

See the *Double-Buffered I/O* section later in this chapter for an explanation of double buffering. Figure 3-23 outlines the basic steps for digital double-buffered group block I/O applications.

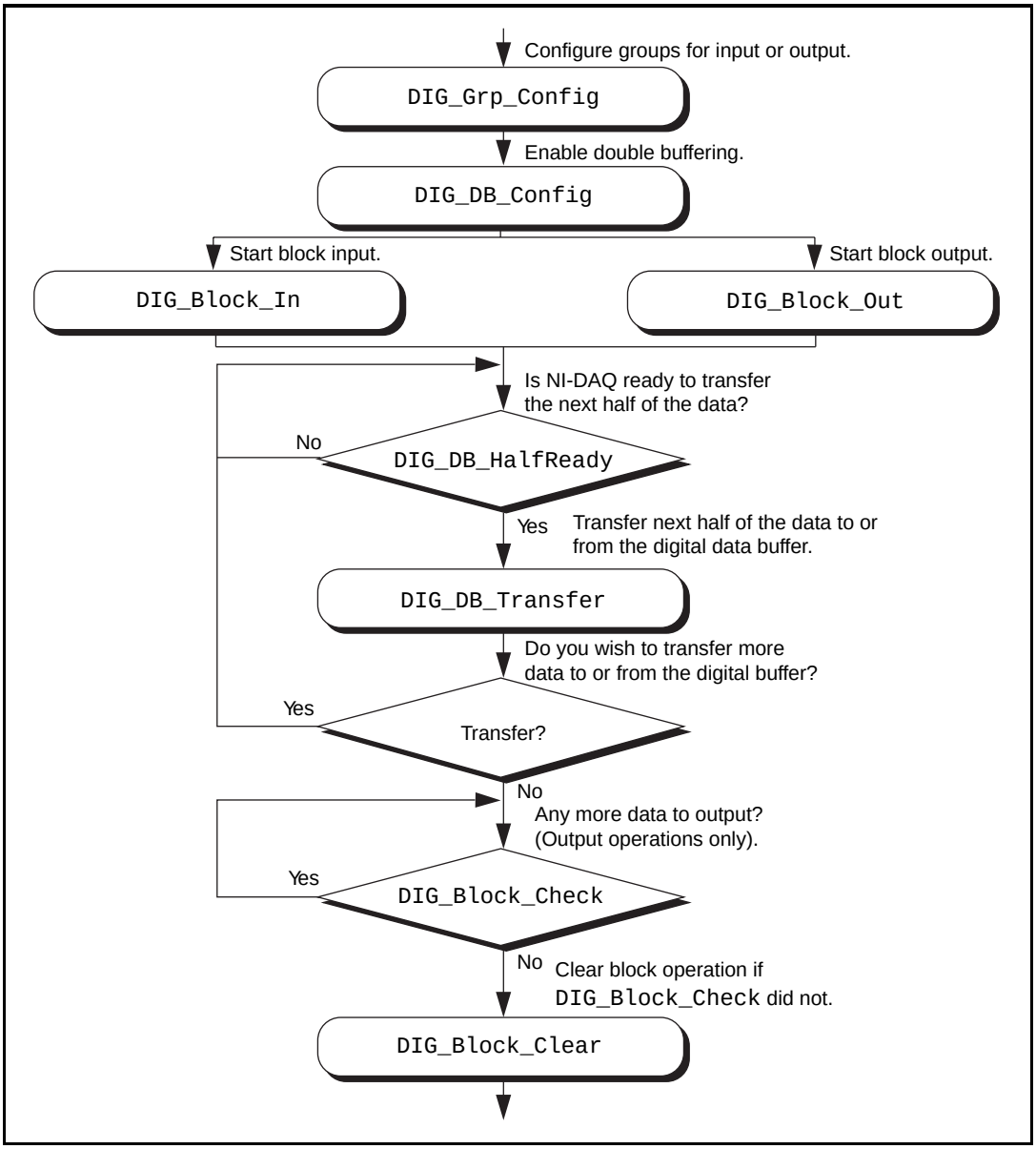


Figure 3-23. Double-Buffered Block Operation

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 3-22, 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 3-23). To start the digital block input or output, call `DIG_Block_In` or `DIG_Block_Out`.

After the operation has started, you can perform any number of transfers to or from the circular buffer. Input operations transfer new data from the digital buffer for storage or processing. Output operations 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 might 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 actually to output the data. In addition, if NI-DAQ completes the block operation prior to a `DIG_Block_Check` call, `DIG_Block_Check` automatically calls `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 and DAQDIO 6533 (DIO-32HS) Devices

Use pattern generation for clocked digital I/O, when you have a port that is written to or read from based on the output of a counter. 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 so that each can run in this mode at different rates simultaneously. 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, NI-DAQ reads the digital ports. 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 and AT-DIO-32HS use 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 unless you use dual DMA).Refer to Chapter 4, *DMA and Programmed I/O Performance Limitations* for more information.

☞ **Note:** ***For the AT-DIO-32F,*** `DIG_Block_In` ***and*** `DIG_Block_Out` ***return a warning 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 the DIO-32F, another option is to 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 small request intervals 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. 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 for PC Compatibles*.
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 can 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 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` 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 resumes 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` 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 enable these options only 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 MIO devices
- PC-TIO-10

Use the Interval Counter (`ICTR`) functions with the following devices:

- 516 devices
- DAQCard-500/700
- Lab and 1200 devices
- LPM devices

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 General-Purpose Counter/Timer Functions

The General-Purpose Counter/Timer (`GPCTR`) functions perform counting timing I/O and timing counter operations on the Am9513-based MIO devices, and the PC-TIO-10:

`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 square wave you want 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 3-24 shows the 16-bit counters available on the Am9513-based MIO devices and PC-TIO-10.

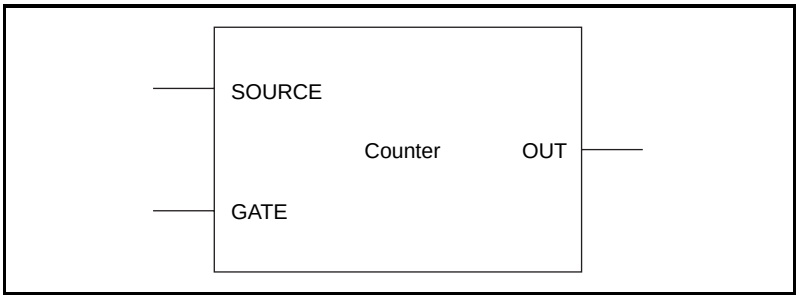


Figure 3-24. Counter Block Diagram

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 MIO-F-5/16X devices 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 3-25 shows examples of the four types of output signals generated.

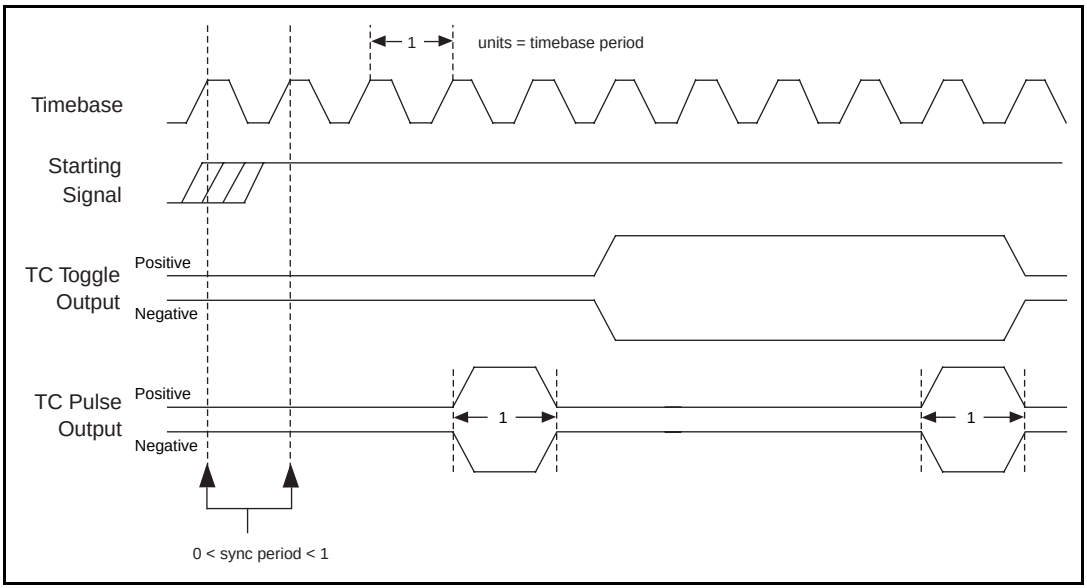


Figure 3-25. Counter Timing and Output Types

Figure 3-25 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 is 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 3-25, represents either a software starting of the counter, for the no-gating case, or some sort of signal at the GATE input. The signal is 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 3-25, 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 MIO 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 PC Compatibles* 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 3-26 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 3-27 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 produces one cycle of the square wave.

Besides `CTR_Square`, you also can 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` only can generate a square wave with a 50 percent 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` simultaneously can start, stop, and restart any number of counters. Also, `CTR_Simul_Op` simultaneously can save all the current counter values

to their hold registers, which you can read later, one at a time. See Figure 3-28 on how to incorporate `CTR_Simul_Op` with other counter functions like `CTR_EvCount` and `CTR_Pulse`.

Am9513-based MIO 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 can be used for waveform generation later 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.

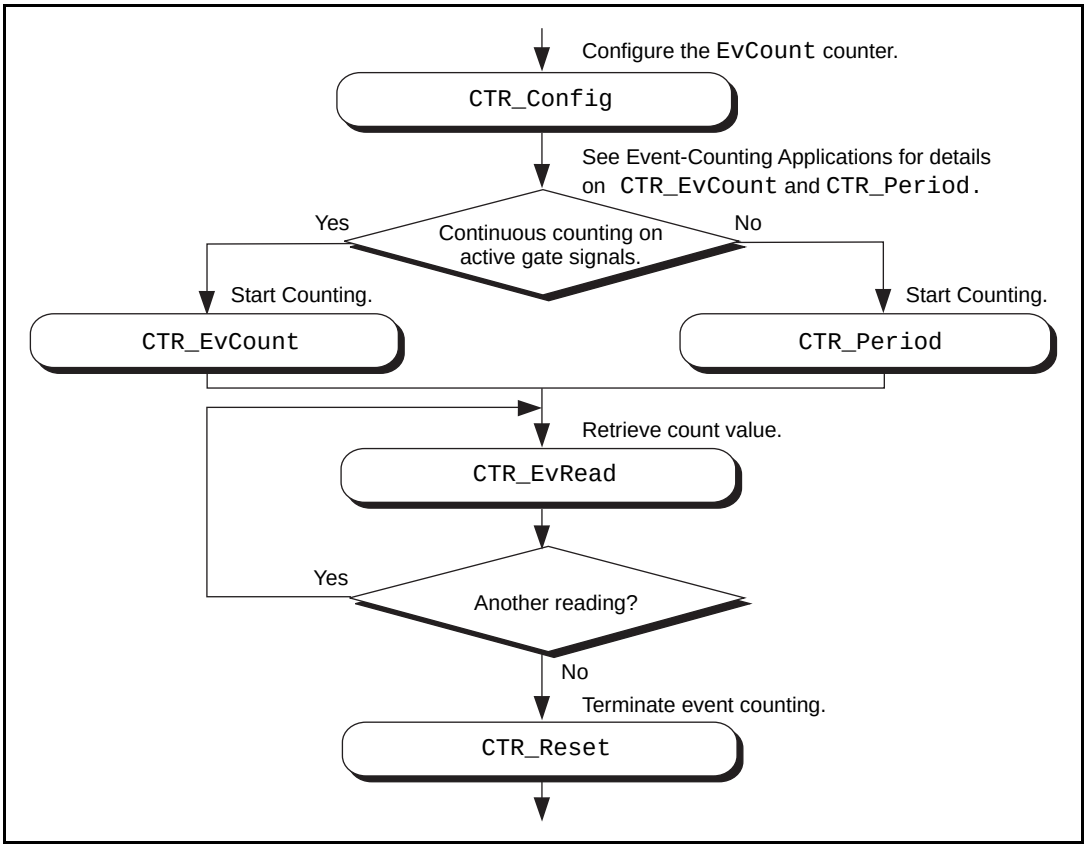


Figure 3-26. Event Counting

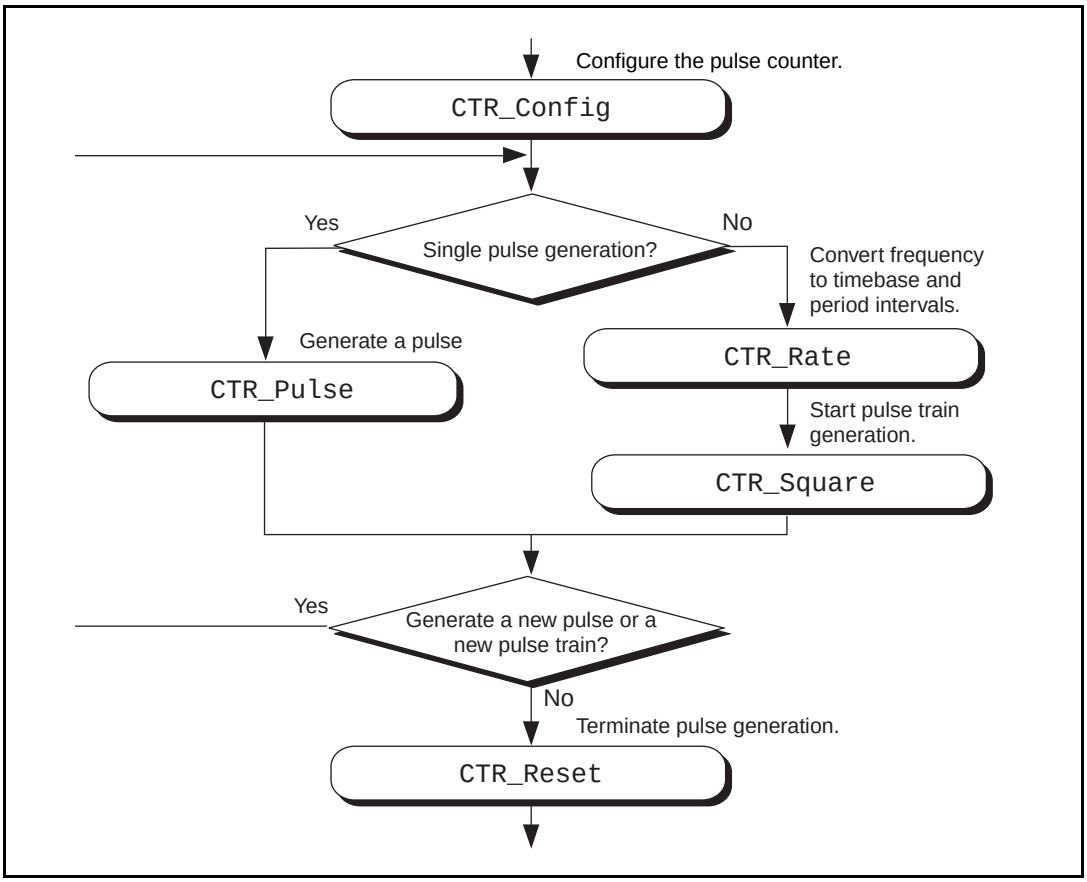


Figure 3-27. Pulse Generation

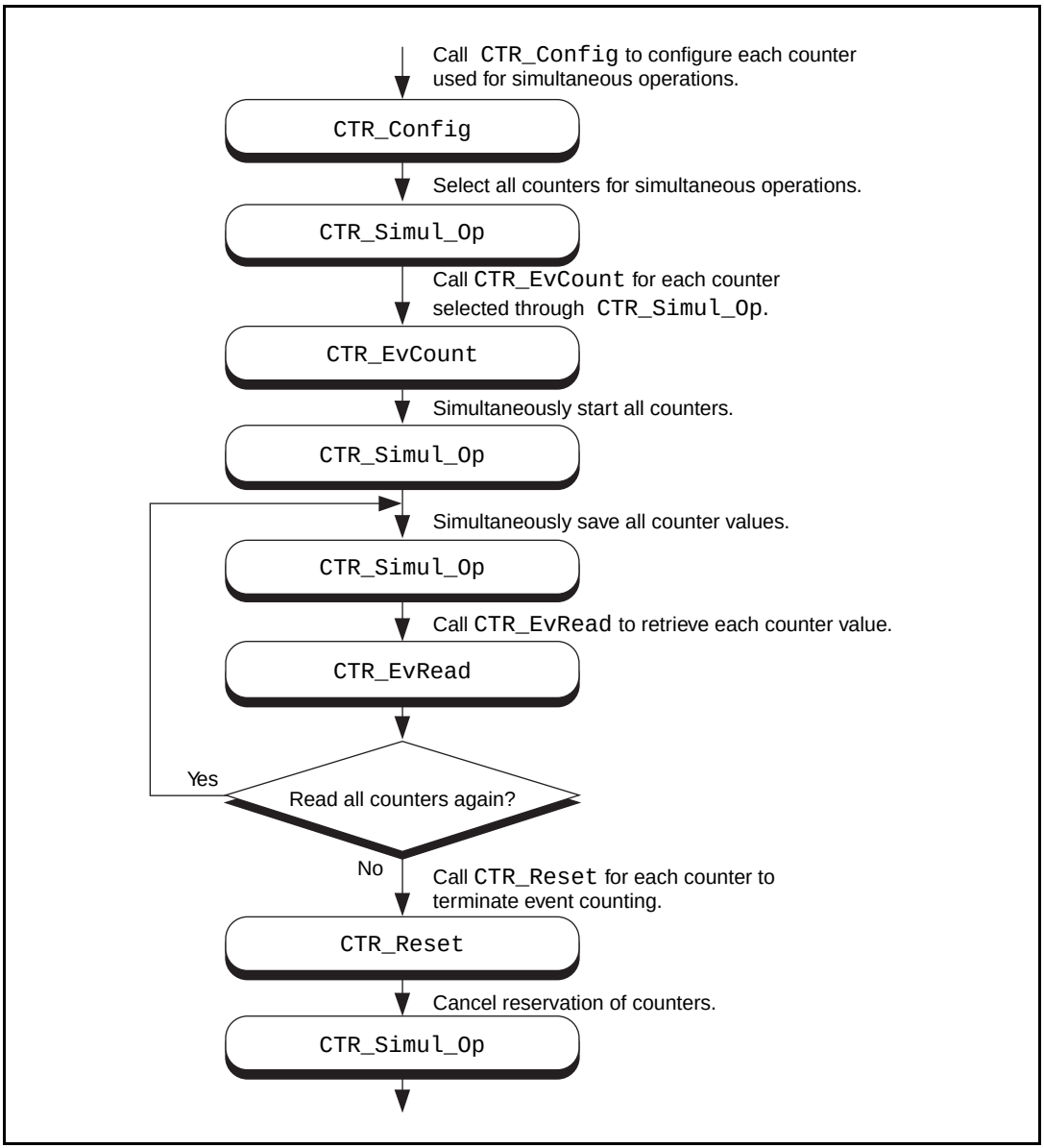


Figure 3-28. Simultaneous Counter Operation

Event-Counting Applications

`CTR_EvCount` and `CTR_EvRead` work with 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 3-29 illustrates timer event counting.

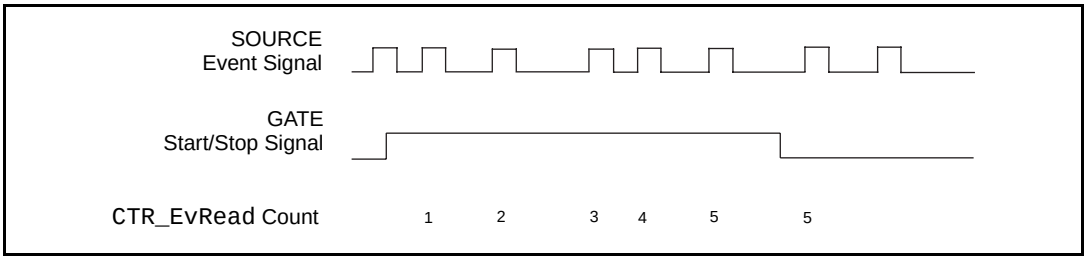


Figure 3-29. Timer Event Counting

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 3-30 shows a typical pulse-width measurement.

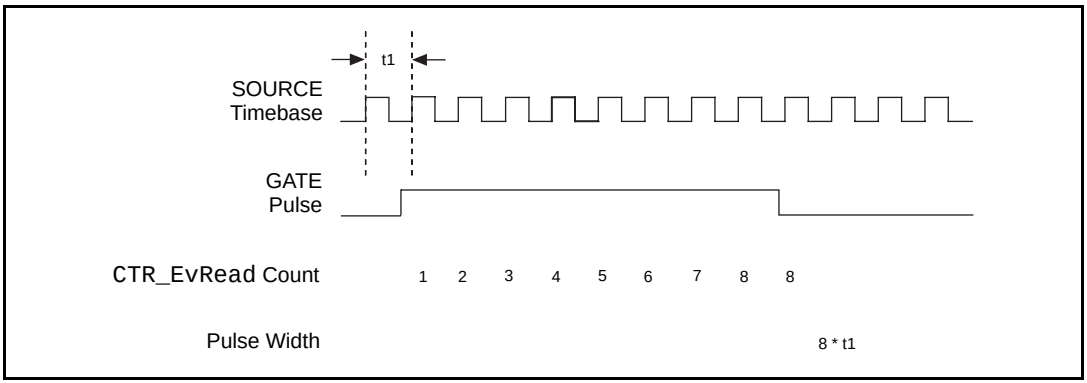


Figure 3-30. Pulse-Width Measurement

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 PC Compatibles* for more information). Figure 3-31 illustrates a frequency measurement.

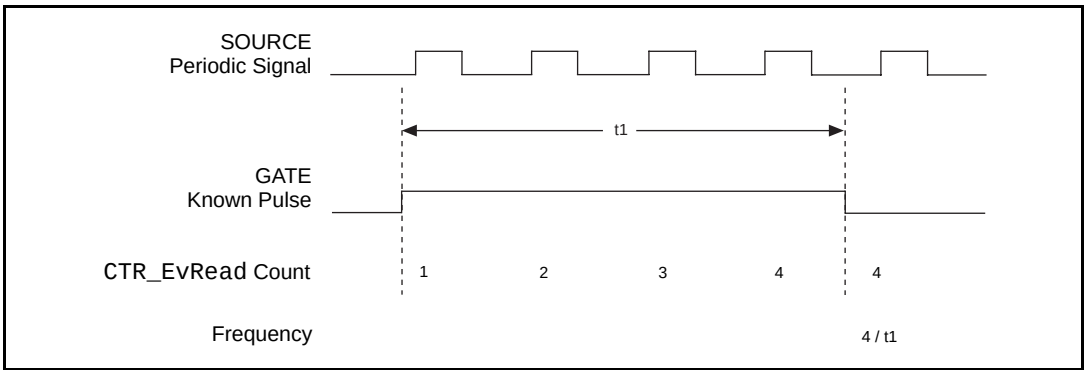


Figure 3-31. Frequency Measurement

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 for PC Compatibles*). 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 when 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 3-31) 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 3-32 shows a continuous period measurement.

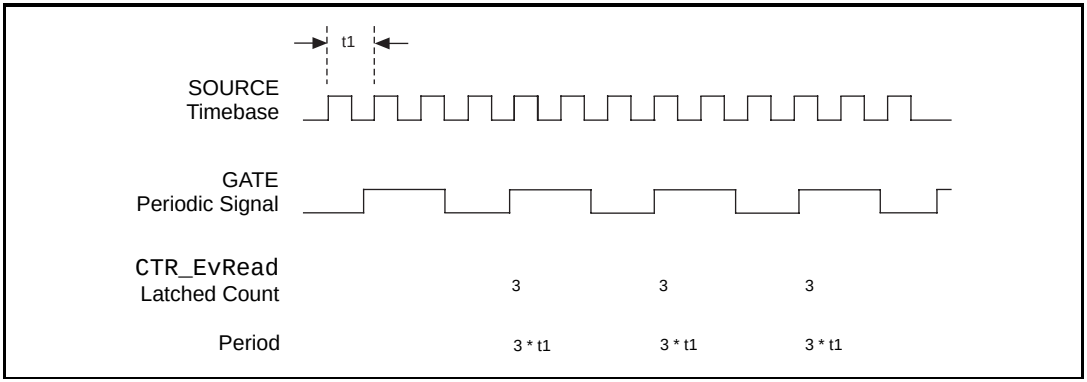


Figure 3-32. Continuous Period Measurement

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 only can 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 then can 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 516 devices, DAQCard-500/700, Lab and 1200 devices, and LPM devices:

`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 assigned counter to operate in the specified mode.

Interval Counter/Timer Operation for the ICTR Functions

Figure 3-33 shows the 16-bit counters on the 516 devices, DAQCard-500/700, Lab and 1200 devices, and LPM devices.

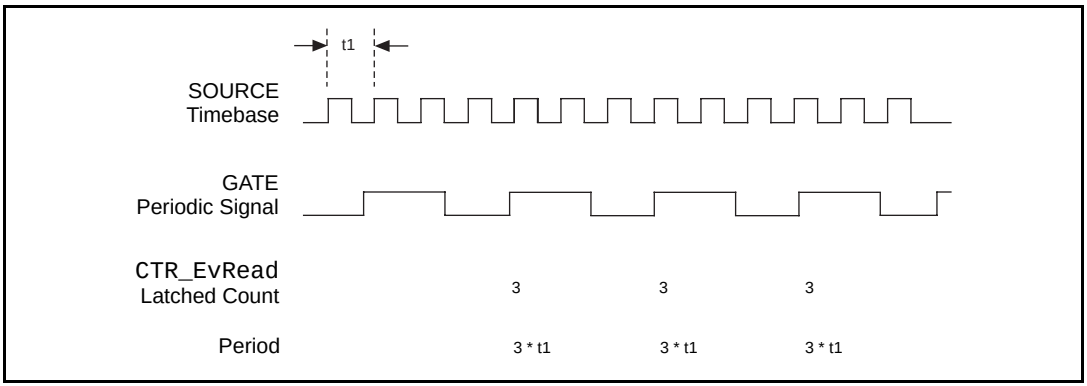


Figure 3-33. Interval Counter Block Diagram

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*, *Lab-PC-1200/AI User Manual*, *PCI-1200 Register-Level Programmer Manual*, and *SCXI-1200 Register-Level Programmer Manual*, and the MSM82C54 data sheet included in the *PC-LPM-16/PnP User Manual*, *DAQCard-500 Register-Level Programmer Manual*, and *DAQCard-700 Register-Level Programmer 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 PC Compatibles* 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 uses for a buffered counter operation.
GPCTR_Control	Controls the operation of the general-purpose counter.
GPCTR_Set_Application	Selects the application for which you uses the general-purpose counter. The function description in the *NI-DAQ Function Reference Manual for PC Compatibles* 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. To learn more about the GPCTR functions, read the introduction to the description of the GPCTR_Set_Application function in Chapter 2, *Function Reference of the NI-DAQ Function Reference Manual for PC Compatibles* and sections that pertain to the applications you want to perform with the counter. You then can refer to descriptions of other GPCTR functions for details.

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 (non-E Series devices only).
`RTSI_Clock`	Connects or disconnects the system clock from the RTSI bus (non-E Series devices only).
`RTSI_Conn`	Connects a device signal to the specified RTSI bus trigger line (non-E Series devices only).
`RTSI_DisConn`	Disconnects a device signal from the specified RTSI bus trigger line (non-E Series devices only).
`Select_Signal`	Connects or disconnects a device signal to a RTSI bus trigger line (E Series devices only).

The following devices have an interface to the RTSI bus trigger lines:

- AT-AO-6/10
- AT-DIO-32F
- AT-DIO-32HS
- 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-10
- AT-MIO-16XE-50
- NEC-AI-16E-4
- NEC-AI-16XE-50
- NEC-MIO-16E-4
- NEC-MIO-16XE-50
- PCI-DIO-32HS
- PCI-MIO-16E-1
- PCI-MIO-16E-4
- PCI-MIO-16XE-10
- PCI-MIO-16XE-50
- PCI-6031E (MIO-64XE-10)
- PCI-6032E (AI-16XE-10)
- PCI-6033E (AI-64XE-10)
- PCI-6071E (MIO-64E-1)
- PXI-6040E
- PXI-6070E
- PXI-6533
- VXI-MIO-64E-1
- VXI-MIO-64XE-10

The RTSI Bus

The RTSI bus is implemented via a 34-pin ribbon cable connector on the AT, NEC, and PCI E Series devices. The RTSI bus is implemented on VXI-DAQ devices using the VXIbus trigger lines. Fourteen of the RTSI bus lines are dedicated to a 7-wire trigger bus. Each device that works with 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 particular 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 outlined 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-16/16D RTSI Connections

The MIO-16/16D devices contain nine 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
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 Function Group* 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*.

MIO-F-5/16X RTSI Connections

The MIO-F-5/16X devices contain nine signals that you can connect to the RTSI bus trigger lines. Table 3-9 shows these signals.

Table 3-9. MIO-F-5/16X 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 Function Group* section earlier in this chapter for more information). GATE1, OUT1, OUT2, OUT5, and FOUT are also available on the I/O connector of the MIO-F-5/16X devices.

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 3-10 shows these signals.

Table 3-10. 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 3-11 shows these signals.

Table 3-11. 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-32 User Manual*.

DAQDIO 6533 (DIO-32HS) RTSI Connections

The AT-DIO-32HS, PCI-DIO-32HS, and PXI-6533 contain eight signals that you can connect to the RTSI bus trigger lines. Table 3-12 shows these signals.

The direction of each signal depends on the function you are performing. Some signals have a different direction when you enable pattern generation using `DIG_Block_PG_Config` than when you leave pattern generation disabled. Make sure that you do not configure a signal as a RTST receiver when you use that signal as a board output. For example, do not configure the DAQDIO 6533 device to receive the

REQ1 line from the RTSI bus if you are using internal requests, or if you have made an external connection that drives the REQ1 pin on the I/O connector.

Table 3-12. DAQDIO 6533 RTSI Bus Signals

Signal Name	Signal Direction (Pattern Direction)	Signal Direction (Handshaking, No Pattern Generation)	Signal Direction (No Handshaking)	Signal Code
REQ1	Receiver (external requests) or source (internal requests)	Receiver	Receiver	0
REQ2	Receiver (external requests) or source (internal requests)	Receiver	Receiver	1
ACK1	Receiver (STARTTRIG1)	Source	Source	2
ACK2	Receiver (STARTRIG2)	Source	Source	3
STOPTRIG1	Receiver	Unused	Receiver	4
STOPTRIG2	Receiver	Unused	Receiver	5
PCLK1	Unused	Source (internal clock) or receiver (external clock)	Source	6
PCLK2	Unused	Source (internal clock) or receiver (external clock)	Source	7

The signals REQ1 and REQ2 are request signals generated internally or received from the I/O connector. ACK1 and ACK2 are acknowledge signals used for handshaking mode; in pattern-generation mode, they can carry start trigger signals instead. PCLK1 and PCLK2 are the peripheral clock lines for burst mode. STOPTRIG1 and STOPTRIG2 are used for data acquisition timing. For more information about the DAQDIO 6533 signals, refer to the *DAQDIO 6533 User Manual*. You can find additional explanations of the signals ACK1, ACK2, STOPTRIG1, and STOPTRIG2 in the `DIG_Trigger_Config` function in Chapter 2, *Function Reference*, in the *NI-DAQ Function Reference Manual for PC Compatibles*.

RTSI Bus Application Hints

This section describes 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.

1. Connect the signals from the device to the RTSI bus.
2. The next step is actually to execute the work of the application.
3. Disconnect the signals from the RTSI bus. Figure 3-34 illustrates the normal order of RTSI function calls.

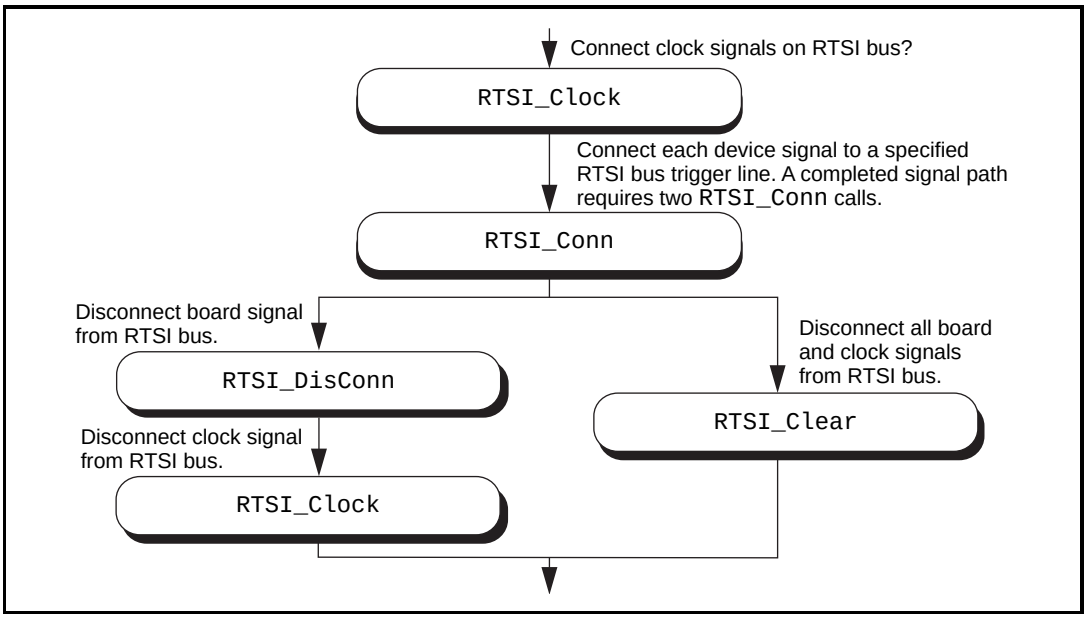


Figure 3-34. Basic RTSI Application Calls

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 receives 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 module to the specified voltage or current output value. You also can 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 particular 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 scale analog input data more accurately 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 also can 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 assigned filter setting on any SCXI module that works with 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 assigned 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_ModuleID_Read`	Reads the Module ID register of the SCXI module in a given slot. The principal difference between this function and `SCXI_Get_Module_Info` is that `SCXI_ModuleID_Read` does a hardware read of the module. You can use this function to verify that your SCXI system is configured and communicating properly.
`SCXI_MuxCtr_Setup`	Enables or disables a DAQ device counter to be used as a multiplexer counter during SCXI channel scanning to synchronize the MIO or AI device, Lab-PC-1200, Lab-PC-1200AI, Lab-PC+, PCI-1200, or SCXI-1200 scan list with the module scan list that NI-DAQ has downloaded to Slot 0 of the SCXI chassis.

`SCXI_Reset`	Resets the specified module to its default state. You also can 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 assigned DAQ device. The function downloads a module scan list to Slot 0 that determines the sequence of modules that are scanned and how many channels on each module are scanned. Each module can 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 works with programmable gain settings (SCXI-1100, SCXI-1102, VXI-SC-1102, SCXI-1122, and SCXI-1141).
`SCXI_Set_Input_Mode`	Configures the SCXI-1122 for differential 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 by disabling the output of any module that was previously driving the SCXIbus. You also can use this function to set up 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 set up 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 3-35 shows the basic structure of an SCXI application.

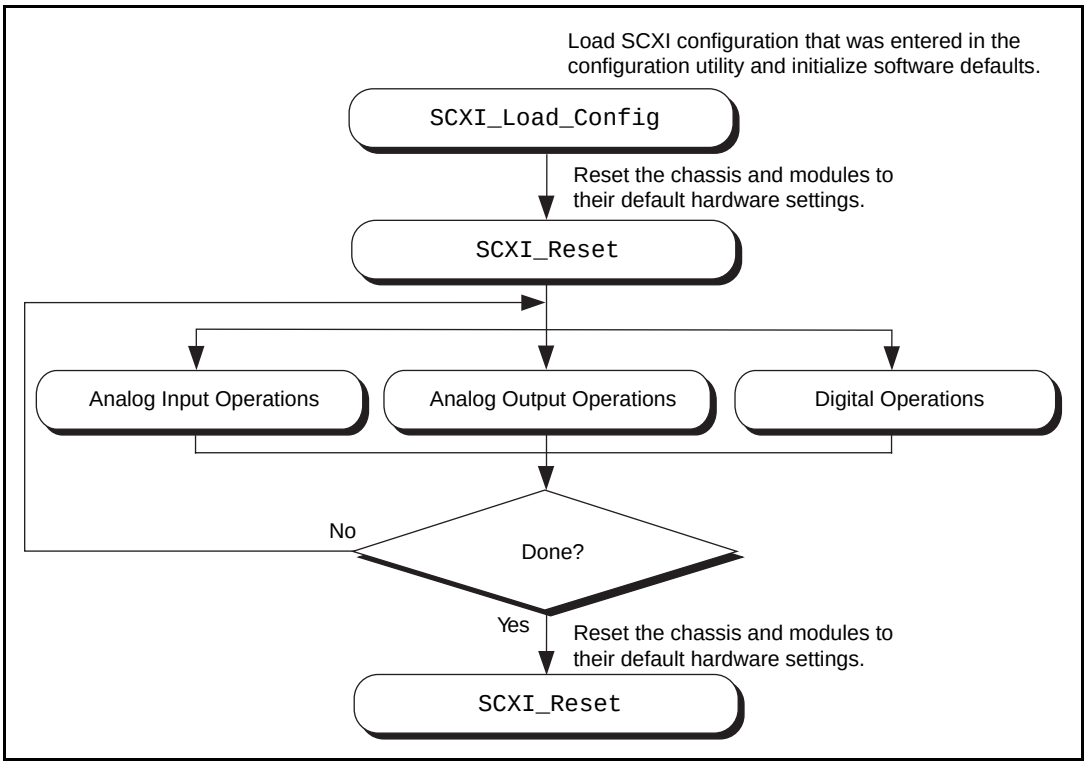


Figure 3-35. General SCXIbus Application

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 3-35. Please refer to the function descriptions in Chapter 2, *Function Reference*, of the *NI-DAQ Function Reference Manual for PC Compatibles* 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 might 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 the multiplexed output of a module is connected by default 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 chassis you want for the operation. You cannot use the SCXI-1200 with multiple chassis.

Figure 3-36 shows the function call sequence of a single-channel or software-scanning application using an SCXI-1100, SCXI-1102, VXI-SC-1102, SCXI-1120, SCXI-1120D, SCXI-1121, SCXI-1122, or SCXI-1141 module operating in Multiplexed mode.

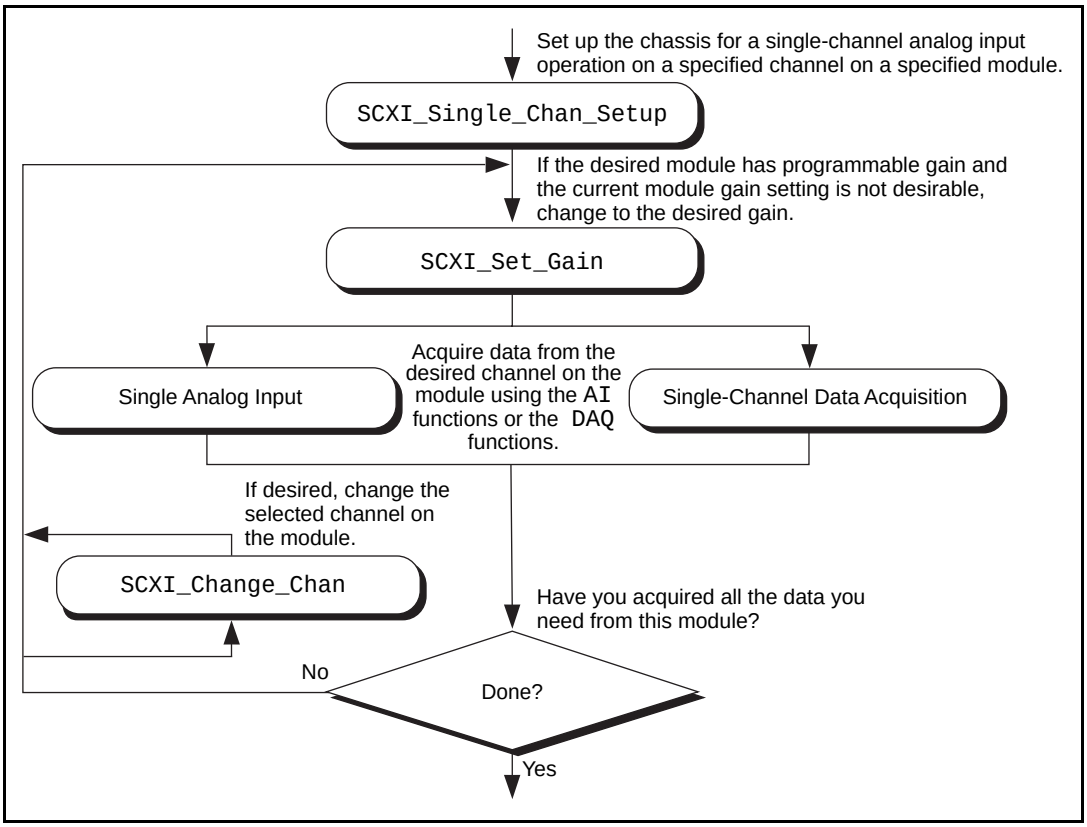


Figure 3-36. Single-Channel or Software-Scanning Operation Using the SCXI-1100, SCXI-1102, VXI-SC-1102, SCXI-1120, SCXI-1120D, SCXI-1121, SCXI-1122, or SCXI-1141 in Multiplexed Mode

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, VXI-SC-1102, SCXI-1122, or SCXI-1141 module. The module maintains this gain setting until you call the function again to change it. You also can 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 channel you want 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 3-37 shows the function call sequence of a single channel or software scanning application using an SCXI-1140 in Multiplexed mode.

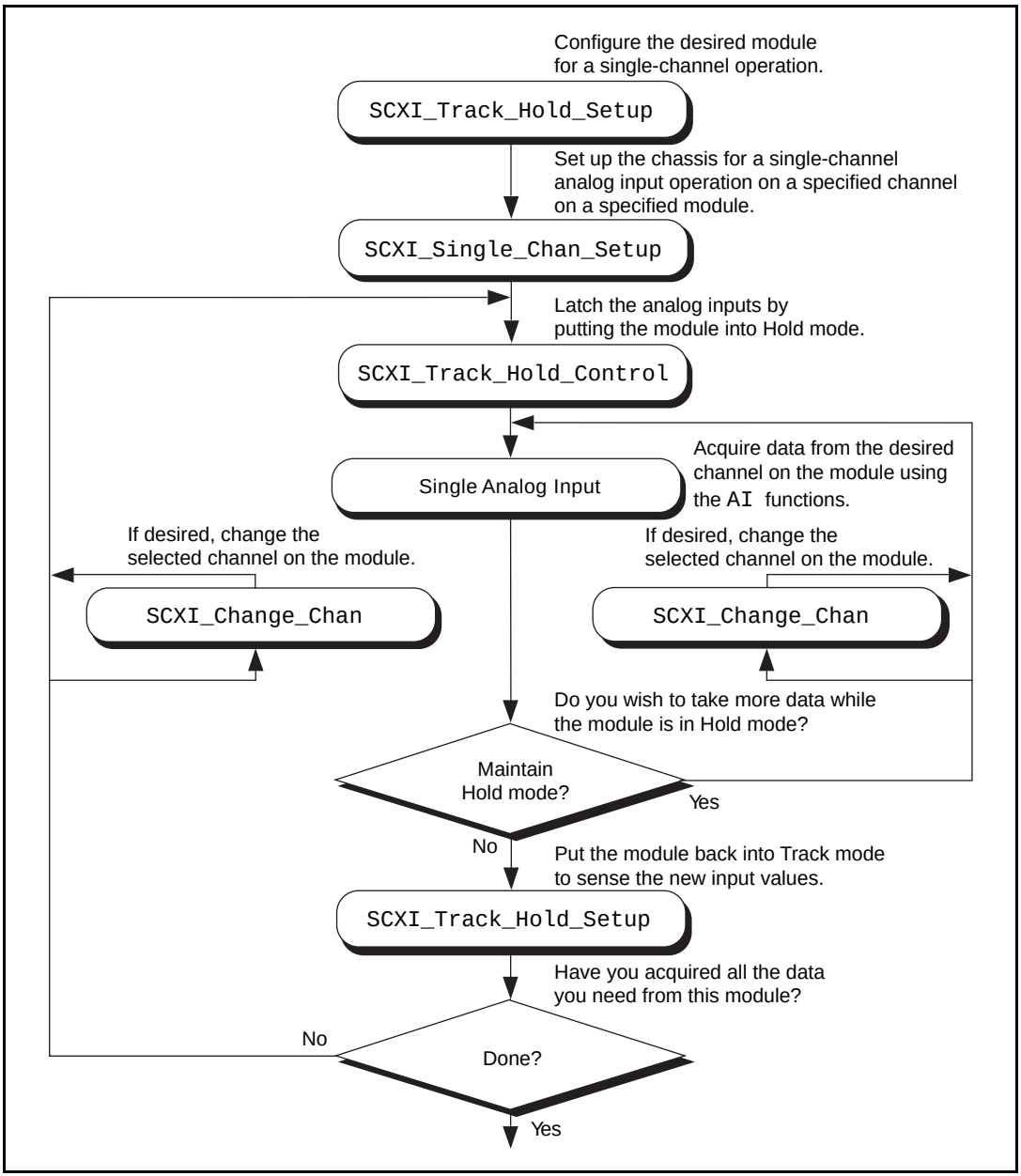


Figure 3-37. Single-Channel or Software-Scanning Operation Using the SCXI-1140 in Multiplexed Mode

Notice the similarities between Figure 3-36 and Figure 3-38, 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 3-38 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 work with 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.

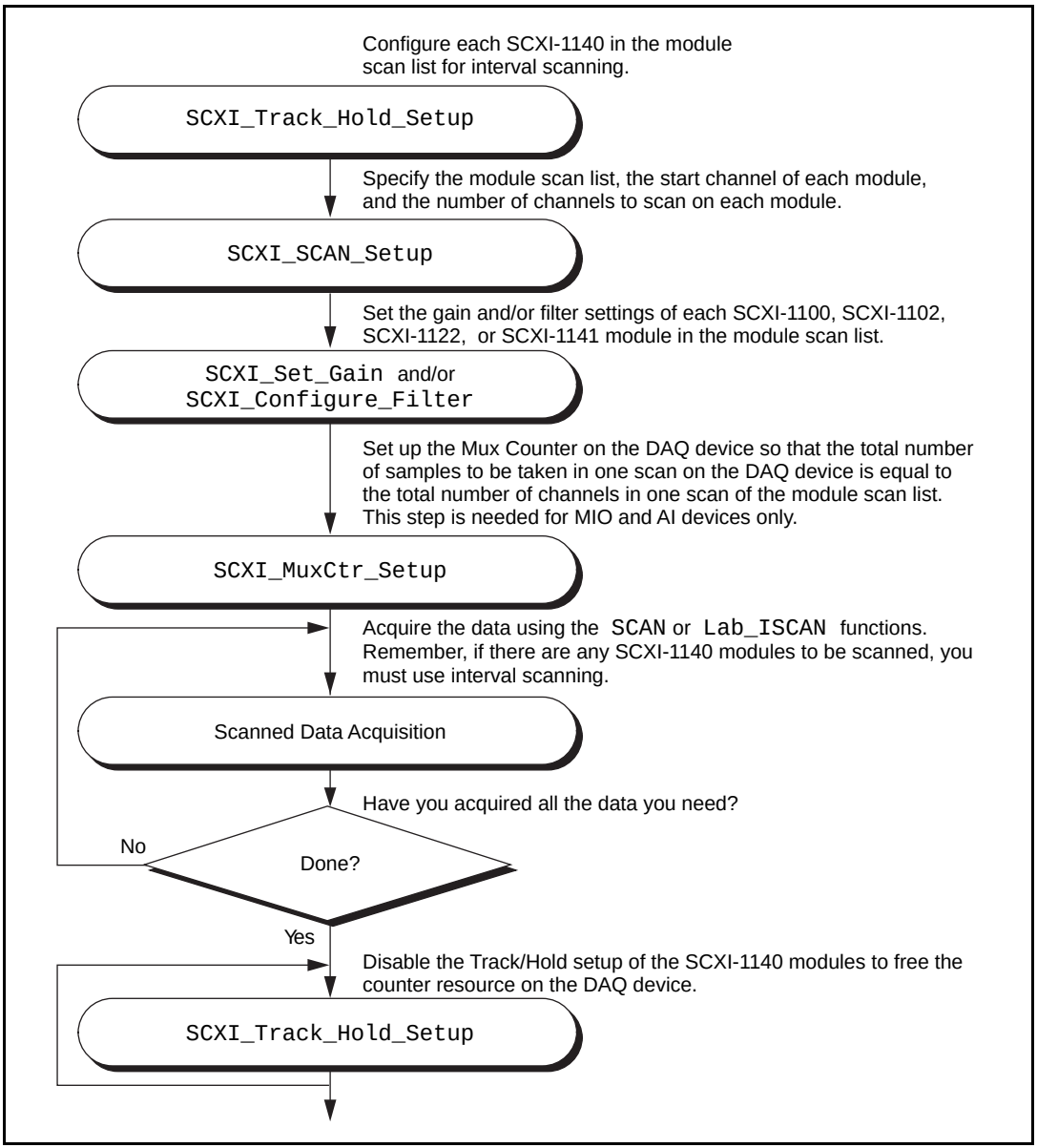


Figure 3-38. Channel-Scanning Operation Using Modules in Multiplexed Mode

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 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 for PC Compatibles*, 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, VXI-SC-1102, SCXI-1122, or SCXI-1141 modules, you can use `SCXI_Set_Gain` to change the gain setting on each module. You also can 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 for PC Compatibles*.

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, pass channel 0 to the `Lab_ISCAN` functions; the SCXI Slot 0 takes care of all the channel switching.

Building Analog Input Applications in Parallel Mode

When you operate the SCXI-1120, SCXI-1120D, SCXI-1121, and SCXI-1141 modules in Parallel mode, no further SCXI function calls are required beyond those shown in Figure 3-38 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 then can 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-1120D, 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 channels on the module you want. 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 for PC Compatibles*, 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, VXI-SC-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 3-39 shows the function call sequence of a single-channel (or software-scanning) operation using the SCXI-1140 module in Parallel mode.

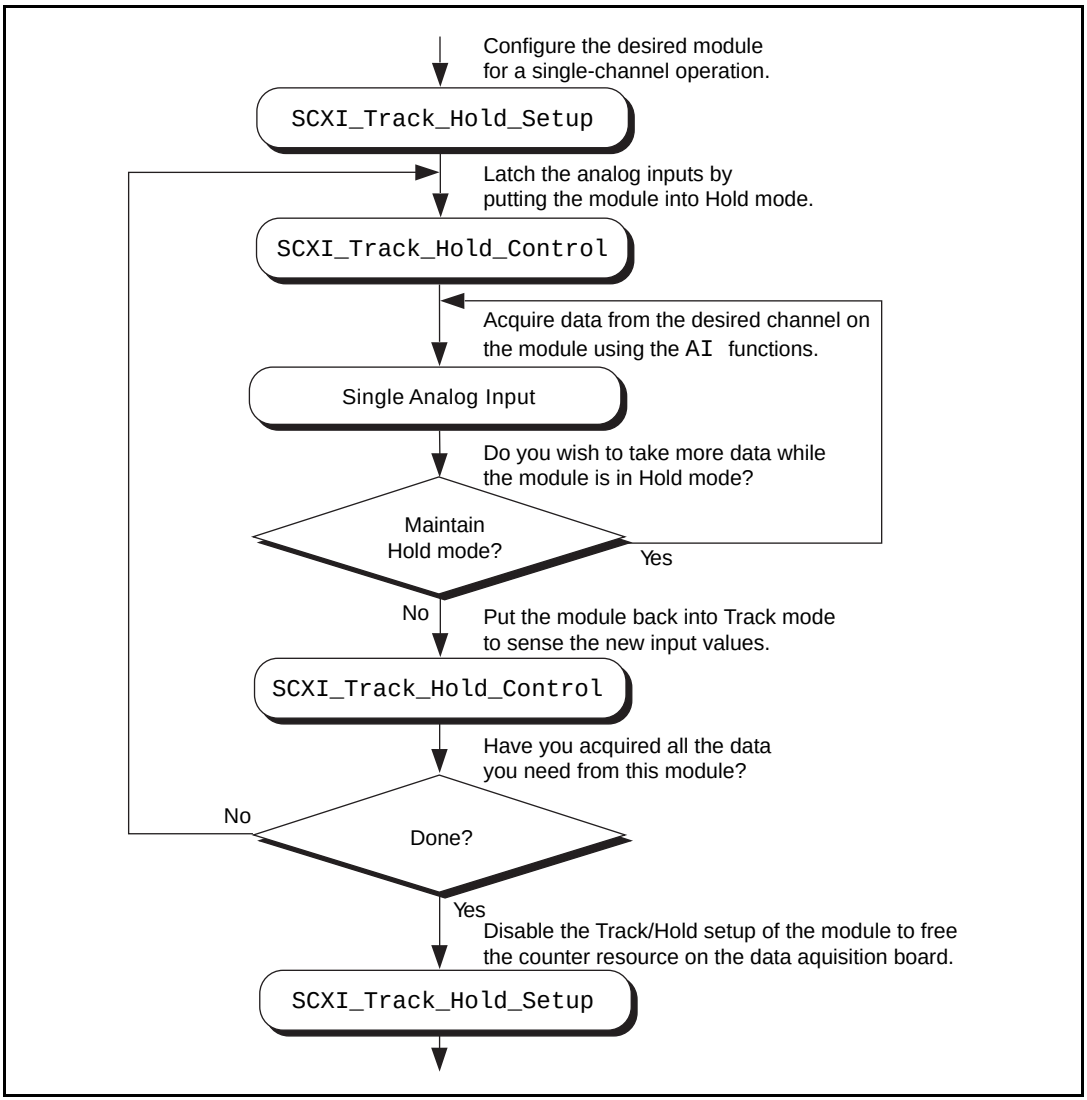


Figure 3-39. Single-Channel or Software-Scanning Operation Using the SCXI-1140 in Parallel Mode

The initial `SCXI_Track_Hold_Setup` call signals the driver that the module is used in a single-channel application, and puts the module into Track mode. The first `SCXI_Track_Hold_Control` call latches, or samples, all the module inputs; subsequent `AI` calls 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 can 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 module channels you want, as described earlier in this section. You also must be aware of the SCXI-1140 Track/Hold timing requirements that were described in *The SCXI-1140* section of Chapter 12, *SCXI Hardware*, of the *DAQ Hardware Overview Guide*.

Figure 3-40 shows the function call sequence of a channel-scanning application using the SCXI-1140 in Parallel mode.

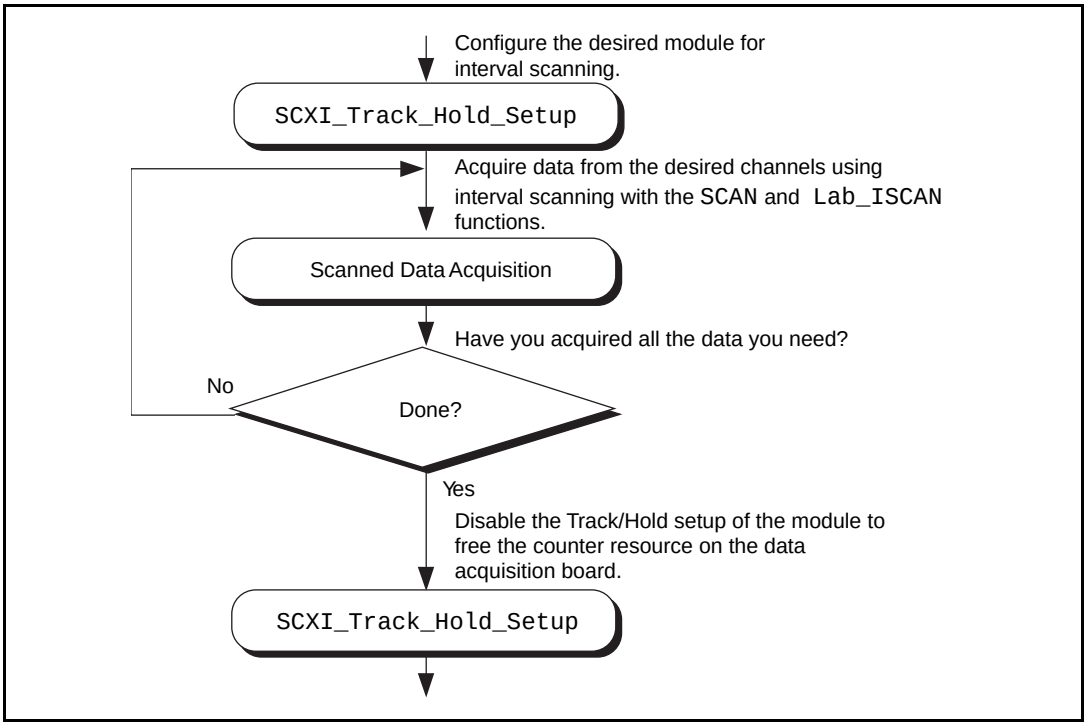


Figure 3-40. Channel-Scanning Operation Using the SCXI-1140 in Parallel Mode

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 and AI devices, Lab-PC+, PCI-1200, SCXI-1200, and DAQCard-1200 devices work with channel-scanning using the SCXI-1140 module.

SCXI Data Acquisition Rates

The settling time of the SCXI modules can affect the maximum data acquisition rates that your DAQ device can achieve. The settling times of the different SCXI modules at each gain setting are listed in Table 3-13 for three different DAQ devices.

The maximum data acquisition rate you can use will be the inverse of the settling time for your SCXI module and DAQ device. For example, if the settling time is listed as 7 µs, your maximum data acquisition rate will be 1/7 µs = 143 kS/s.

If you are using a DAQ device with a maximum acquisition rate faster than the AT-MIO-16E-2 (such as the AT-MIO-16E-1), you should use the settling times and corresponding maximum acquisition rates listed for the AT-MIO-16E-2.

If you are using a DAQ device with a maximum acquisition rate slower than the AT-MIO-16F-5 (such as the AT-MIO-16E-10), you should add 1 µs to the settling time of your DAQ device. The maximum acquisition rate for the AT-MIO-16E-10 would be 1/(10 µs + 1 µs) = 90.9 kS/s.

If you are using a DAQ device faster than the AT-MIO-16F-5 but slower than the AT-MIO-16E-2 (such as an AT-MIO-64E-3), you can interpolate between the settling times listed for these devices to calculate an appropriate settling time and corresponding maximum data acquisition rate.

Table 3-13. Maximum SCXI Module Settling Times

SCXI Module	Gain	Settling Time Using AT-MIO-16F-5[1]	Settling Time Using AT-MIO-16E-2 [2]	Settling Time Using AT-MIO-16X (±0.006% Accuracy) [3]	Settling Time Using AT-MIO-16X (±0.0015% Accuracy) [3]
SCXI-1100 (no filter)	1 to 100	7 µs	4 µs	10 µs	32 µs
	200	10 µs	5.5 µs	10 µs	33 µs
	500	16 µs	12 µs	25 µs	40 µs
	1,000	50 µs	20 µs	30 µs	76 µs
	2,000	50 µs	25 µs	30 µs	195 µs
SCXI-1102, VXI-SC-1102	all gains	7 µs	3 µs	10 µs	
SCXI-1120, SCXI-1120D	all gains	7 µs	3 µs	10 µs	20 µs
SCXI-1121	all gains	7 µs	3 µs	10 µs	20 µs
SCXI-1122	all gains	10 ms	10 ms	10 ms	10 ms
SCXI-1140	all gains	7 µs	3 µs	10 µs	20 µs
SCXI-1141	all gains	7 µs	3 µs	10 µs	20 µs

[1] Includes effects of AT-MIO-16F-5 with 1 m SCXI cable assembly.

[2] Includes effects of AT-MIO-16E-2 with 1 m or 2 m SCXI cable assembly.

[3] Includes effects of AT-MIO-16X with 1 m or 2 m SCXI cable assembly.

Note: If you are using remote SCXI, the maximum data acquisition rate also depends on the serial baud rate used. For more information, see the *SCXI Chassis User Manual*.

Table 3-14. SCXI-1200 Module Settling Rates

Gain	Maximum Acquisition Rate	Settling Time
1	83.3 kS/s	12 µs
2 to 50	55 kS/s	18 µs
100	25 kS/s	40 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 dramatically affects 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 3-39 instead of the channel-scanning method shown in Figure 3-40. 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 increases the lifetime of your module relays. When 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 the voltages you want to the DAC channels on the module. You can use the `SCXI_Get_Status` function, if you want, 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 might want to use the `SCXI_Get_Status` function after calling the `SCXI_Set_State` function. `SCXI_Get_Status` tells you when the SCXI-1160 relays have finished switching.

If you are using the SCXI-1162 or SCXI-1162HV module, `SCXI_Get_State` reads the module input channels. For the other digital and relay modules, `SCXI_Get_State` returns 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` reads 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. Chapter 7, *DIO-96 Digital I/O Boards*, Chapter 8, *DIO-24, AT-MIO-16D and AT-MIO-16DE-10 Devices*, and Chapter 9, *DIO-32F and DAQDIO 6533 (DIO-32HS) Digital I/O Boards*, in the *DAQ Hardware Overview Guide*, 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 units for 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 B-, E-, J-, K-, N-, R-, S-, or T-type thermocouples into temperature in Celsius, Fahrenheit, kelvin, or Rankine.

NI-DAQ 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.

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 make any necessary editing easier.

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
and
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 contains 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 = 3.9784×10^{-3}

  B = -5.8408×10^{-7}

  ($\alpha = 3.92 \times 10^{-3}$; $\partial = 1.492$)

Some RTD documentation contains values for α and ∂, from which you can calculate **A** and **B** using the following equations:

$$\mathbf{A} = \alpha(1 + \partial/100)$$

$$\mathbf{B} = -\alpha\ \partial/10{,}000100^2$$

where α is the temperature coefficient at T = 0° C.

$$\mathbf{C} = -\alpha\ ß/1{,}000{,}000$$

where ß is a characteristic of your RTD similar to the α and ∂ equation coefficients.

Strain_Convert and 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 3-41 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 (needed only 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.

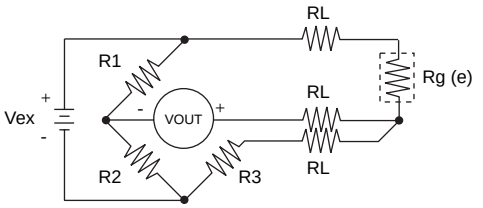
The `Strain_Buf_Convert` routine has three remaining parameters—**numPts** is the number of voltage points to convert, **strainVoltBuf** is the array that contains the voltages that NI-DAQ read from the strain gauge, and **strainValBuf** is the return array that contains the strain values.

Using This Function

The conversion formula used is based solely on the bridge configuration. Figure 3-41 shows the seven bridge configurations supported and the corresponding formulas. For all bridge configurations, NI-DAQ uses the following formula to obtain Vr:

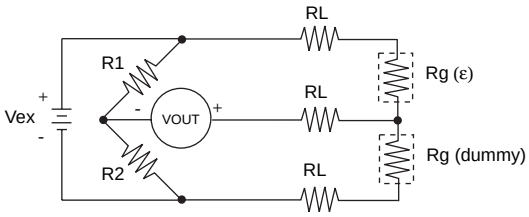
Vr = (**strainVolts - Vinit**) / **Vex**

In the circuit diagrams shown in Figure 3-41, V_{OUT} 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.



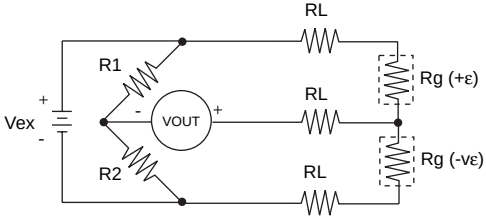
bridgeConfig = 1 (QTR_BRIDGE_I)

$$\text{strain } (\varepsilon) = \frac{-4Vr}{GF\,(1 + 2Vr)} \cdot \left(1 + \frac{RL}{Rg}\right)$$



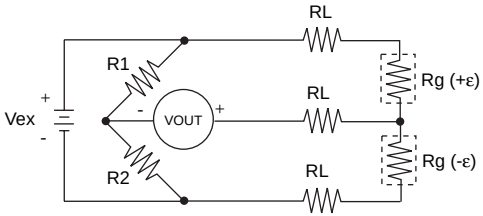
bridgeConfig = 2 (QTR_BRIDGE_II)

$$\text{strain } (\varepsilon) = \frac{-4Vr}{GF\,(1 + 2Vr)} \cdot \left(1 + \frac{RL}{Rg}\right)$$



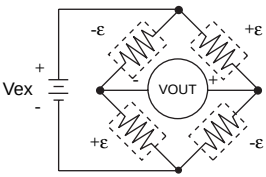
bridgeConfig = 3 (HALF_BRIDGE_I)

$$\text{strain } (\varepsilon) = \frac{-4Vr}{GF\,[(1 + \nu) - 2Vr\,(\nu - 1)]} \cdot \left(1 + \frac{RL}{Rg}\right)$$



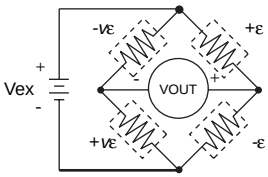
bridgeConfig = 4 (HALF_BRIDGE_II)

$$\text{strain } (\varepsilon) = \frac{-2Vr}{GF} \cdot \left(1 + \frac{RL}{Rg}\right)$$



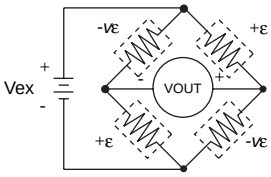
bridgeConfig = 5 (FULL_BRIDGE_I)

$$\text{strain } (\varepsilon) = \frac{-Vr}{GF}$$



bridgeConfig = 6 (FULL_BRIDGE_II)

$$\text{strain } (\varepsilon) = \frac{-2Vr}{GF\,(\nu + 1)}$$



bridgeConfig = 7 (FULL_BRIDGE_III)

$$\text{strain } (\varepsilon) = \frac{-2Vr}{GF\,[(\nu + 1) - Vr\,(\nu - 1)]}$$

Figure 3-41. Strain Gauge Bridge Configuration

Thermistor_Convert and 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 3-42). 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 3-42). 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 assigned 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 assigned 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 assigned in units determined by **TempScale**.

Using This Function

The following equation expresses the relationship between **Volts** and **Rt**, the thermistor resistance (see Figure 3-42).

Volts = **Vref** (**Rt** / (**R1** + **Rt**))

Solving the previous equation for **Rt**, we have:

Rt = **R1** (**Volts** / (**Vref** + **Volts**))

Once this function calculates **Rt**, the function uses the following equation to convert **Rt**, the thermistor resistance, to temperature in kelvin. The function then converts the temperature to the temperature scale you want, if necessary.

T = 1 / (**a** + **b**(ln **Rt**) + **c**(ln **Rt**)^3)

The values used for ***a***, ***b***, and ***c*** are given below. If you are using a thermistor with different values for ***a***, ***b***, and ***c*** (consult your thermistor data sheet), you can edit the thermistor conversion routine to use your own ***a***, ***b***, and ***c*** values.

a = 1.295361E-3

The following equation expresses the relationship between **Volts** and `Rt`, the thermistor resistance (see Figure 3-42).

Volts = **Vref** (`Rt` / (**R1** + `Rt`))

Solving the previous equation for `Rt`, you have:

`Rt` = **R1** (**Volts** / (**Vref** - **Volts**))

When you calculate `Rt`, you use the following equation to convert `Rt`, the thermistor resistance, to temperature in kelvin. We then convert the temperature to the temperature scale you want, if necessary.

T = 1 / (`a` + `b`(ln `Rt`) + `c`(ln `Rt`)^3)

The values used for `a`, `b`, and `c` are shown 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

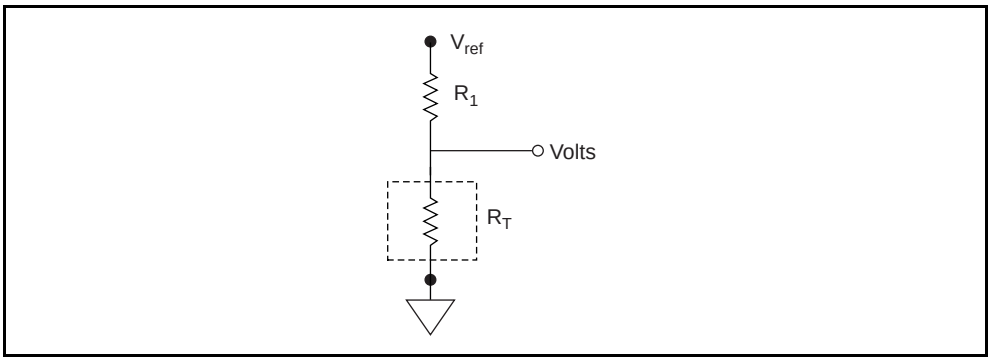


Figure 3-42. Circuit Diagram of a Thermistor in a Voltage Divider

Thermocouple_Convert and 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 shown 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 uses for cold-junction compensation of the thermocouple temperature. If you are using SCXI, this most likely is 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 shown 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 contains 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 3-15. All the reference equations and inverse equations used in these routines are from *NIST Monograph 175*.

Table 3-15 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 3-15. 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
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`, 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 Windows applications.

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 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 can 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 a DIO-32, refer to the `Set_DAQ_Device_Info` function description in Chapter 2, *Function Reference*, of the *NI-DAQ Function Reference Manual for PC Compatibles*, to find out the transfer mode you are 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+, PCI-1200, SCXI-1200, DAQPad-1200, and DAQCard-1200
- DIO-24
- DIO-96
- DAQDIO 6533 (DIO-32HS)

The following boards use DMA for block digital input and output:

- AT-DIO-32F
- DAQDIO 6533 (DIO-32HS)

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 allocate 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 or AT-DIO-32HS, and you have selected the transfer mode so that group 1 and group 2 each has one DMA channel available for digital input/output. The problem is less severe in the case of the AT-DIO-32HS because this device has a small FIFO buffer.

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 **memPageError** warning is returned.

When the DIO-32F is 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, AT-DIO-32F, or AT-DIO-32HS:

- 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.
- If you are using an AT-MIO-16F-5 or AT-DIO-32F, 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.

- 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.
- If you are using an AT-DIO-32HS and want to optimize one group for high-speed transfers, set the transfer mode for that group so the group uses both DMA channels. If you use the transfer mode for the other group, set it to interrupt mode.

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 uses an intermediate (mirror) buffer to transfer data to and from the DAQ device and then copy data to the user buffer above 16 MB.

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. In Windows, this happens at Windows startup time, and in 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 a **memLockError** 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 might 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 when 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 Windows

Interrupt latency in Windows can impose performance limitations on data acquisition. The magnitude of the performance reduction depends on the board and the method used to acquire the data (programmed I/O versus DMA).

Interrupt latency is the delay between the time hardware asserts an interrupt and when the interrupt service routine is activated. In Windows, system software transfers control to the interrupt service routine, imposing a software delay. 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.

DMA Performance in Windows

Buffers Requiring Reprogramming

Interrupt latency can slow data acquisition that uses DMA when DMA reprogramming is required. NI-DAQ might have to reprogram the DMA controller for four reasons:

- The DMA controllers for AT bus computers organize memory addresses into 64 KB 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 might 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 might be required because of the virtual memory management system used in Windows. When a buffer is locked into physical memory in preparation for a data acquisition operation, the buffer might 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 might 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.

Results of Performance Limitation

DMA performance limitations manifest themselves in different ways.

- DMA reprogramming for analog input and digital pattern generation input might cause a FIFO overflow error. FIFO overflows occur when an 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 and digital pattern generation output might cause update errors. Update errors occur when the output board cannot update the output data 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 full 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.

- The AT-DIO-32F does not detect overflow or update errors; NI-DAQ cannot explicitly warn you when DMA reprogramming is causing a performance problem during pattern generation with these boards. For this reason, the `DIG_Block_In` and `DIG_Block_Out` functions return a warning (**memPageError**) when DMA reprogramming is required.

Methods for Eliminating Performance Limitations

You can eliminate performance limitations caused by DMA reprogramming several ways:

- A FIFO on a DAQ device can buffer data during reprogramming. However, high-speed acquisitions easily can overflow a small onboard buffer during reprogramming if the interrupt latency is significant.
- 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 are double-buffered digital I/O, and analog waveform generation with partial transfers enabled).
- NI-DAQ 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 or AT-DIO-32HS, you can set the transfer mode so that both DMA channels are available to a single group. In the case of the AT-DIO-32F, this group must be group 1.

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 device 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 device 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, 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 create. You can process or store the data in the transfer buffer however you choose.

Figure 5-1 illustrates a series of sequential data transfers.

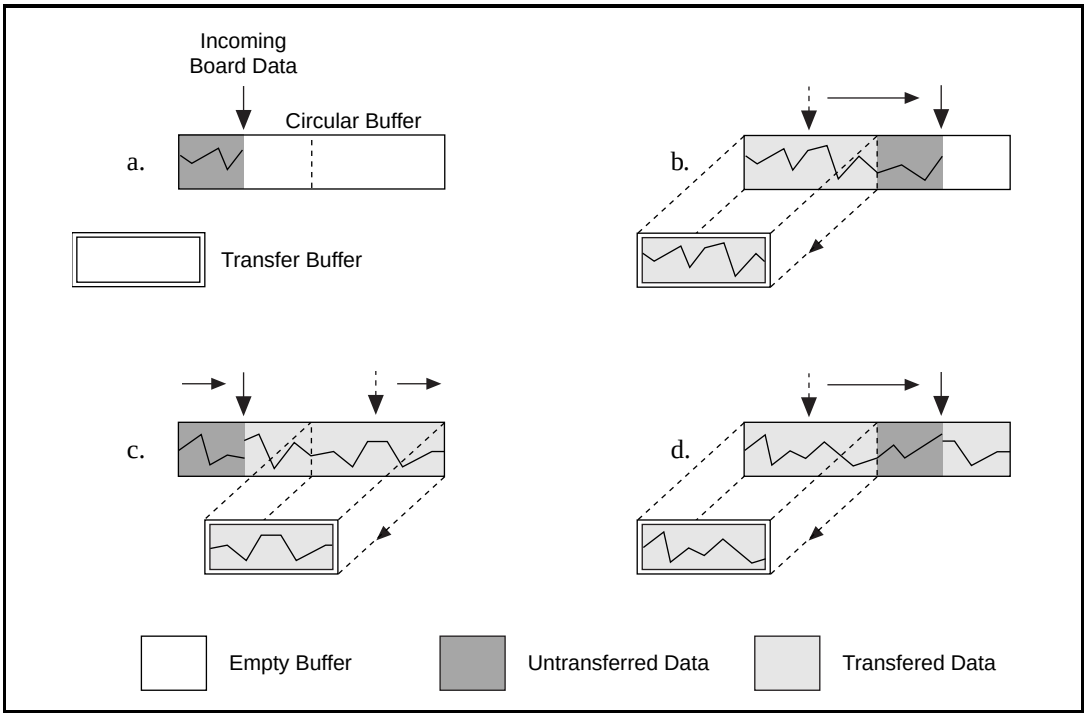


Figure 5-1. Double-Buffered Input with Sequential Data Transfers

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 device 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 device has filled the second half of the circular buffer, the device 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 produce 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.

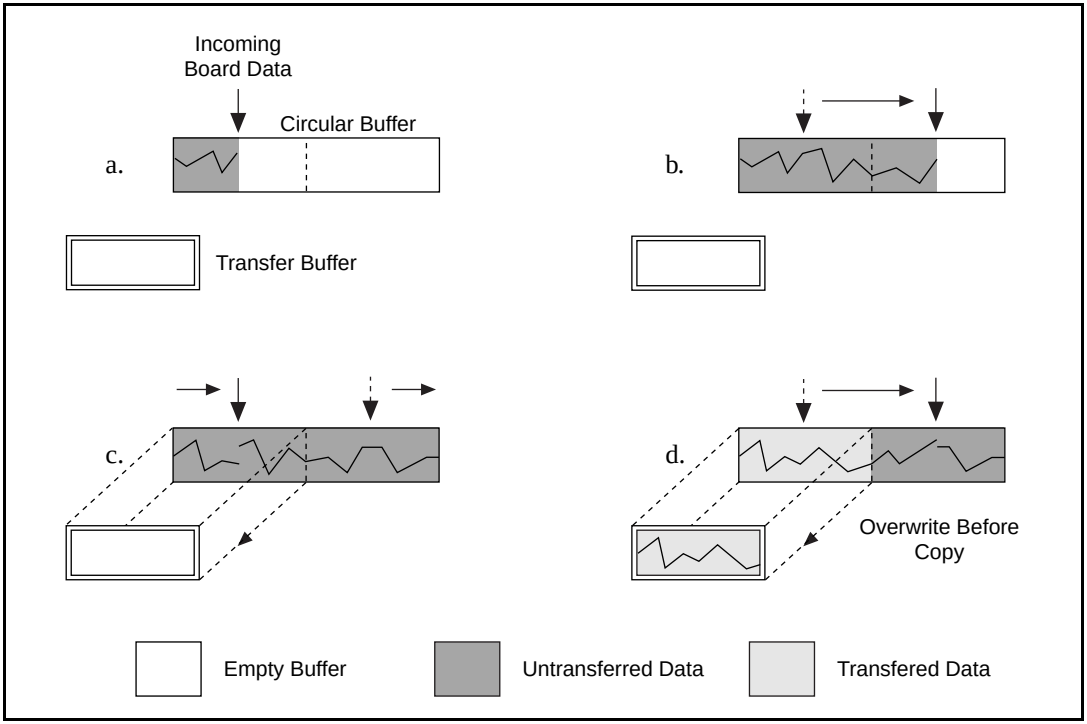


Figure 5-2. Double-Buffered Input with an Overwrite Before Copy

Notice that, 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 device finishes overwriting data in the first half before copying the data into the transfer buffer. After the device 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 (**overWriteError**). 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 device overwrites data that NI-DAQ is simultaneously copying to the transfer buffer. NI-DAQ returns an overwrite error (**overWriteError**) when this occurs. The situation is presented in Figure 5-3.

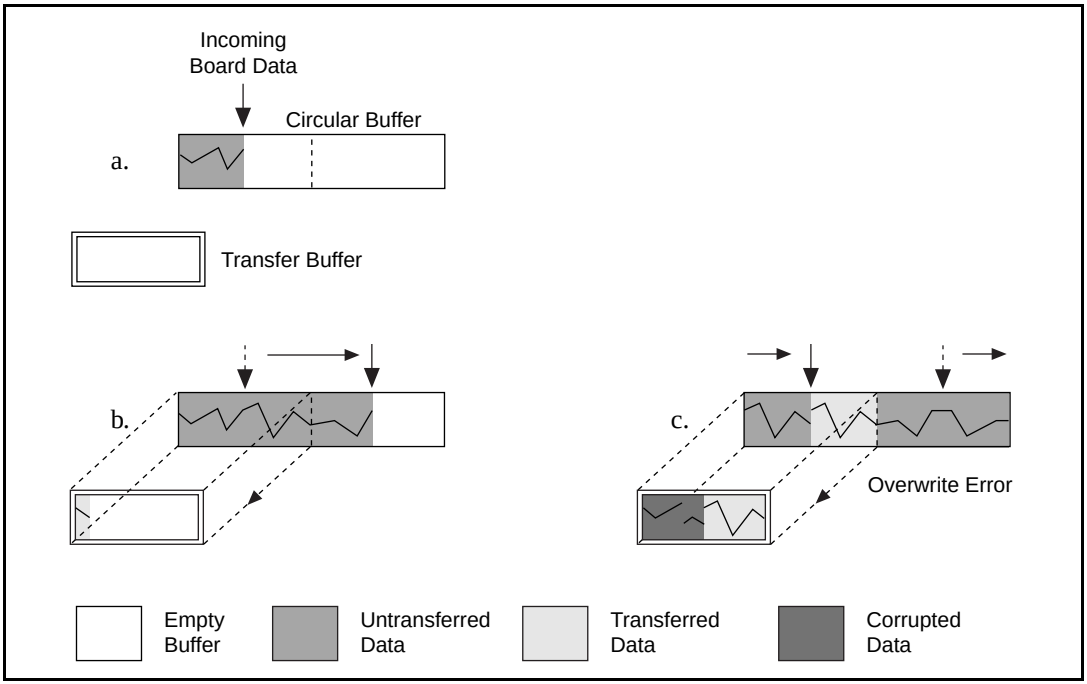


Figure 5-3. Double-Buffered Input with an Overwrite

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 might be corrupted; that is, it might 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 create 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.

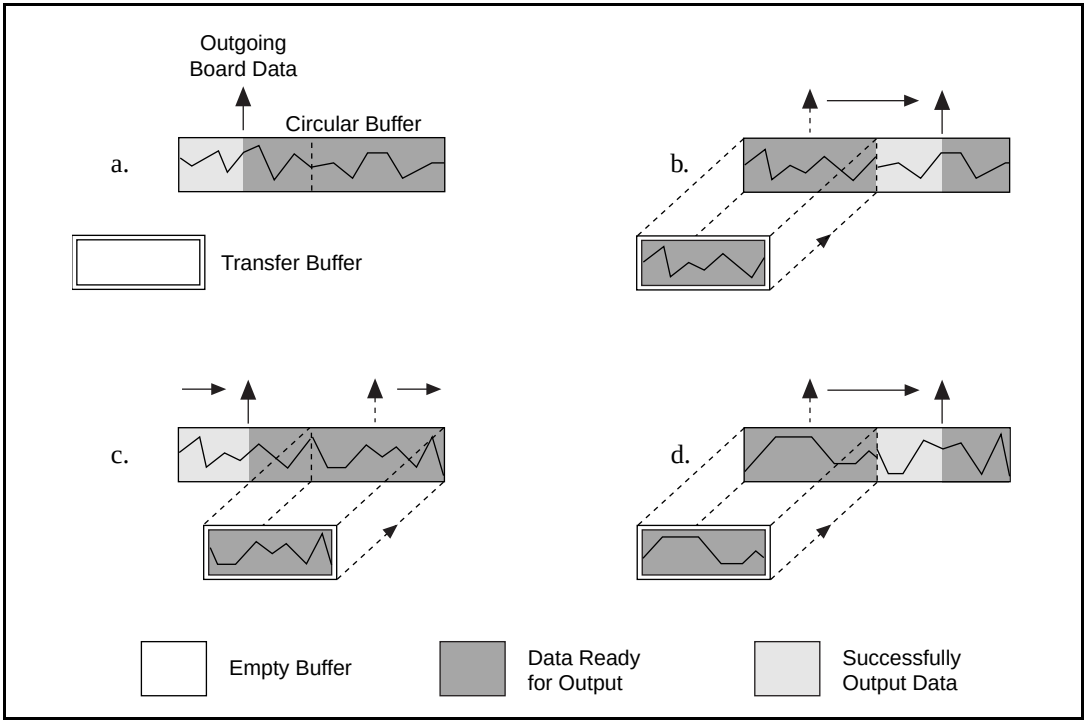


Figure 5-4. Double-Buffered Output with Sequential Data Transfers

The double-buffered output operation begins when the output device begins outputting data from the first half of the circular buffer (Figure 5-4a). After the device 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 then can be updated with new data by your

application. After the output device has finished with the second half of the circular buffer, the device returns to the first half buffer and begins outputting updated data from the first half. NI-DAQ now can 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 produce 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 undergo two potential problems. The first is the possibility of the output device 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.

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 device is retrieving data from the second half. As a result, the device 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 device finishes retrieving data from the first half before copying the data from the transfer buffer. After the device 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).

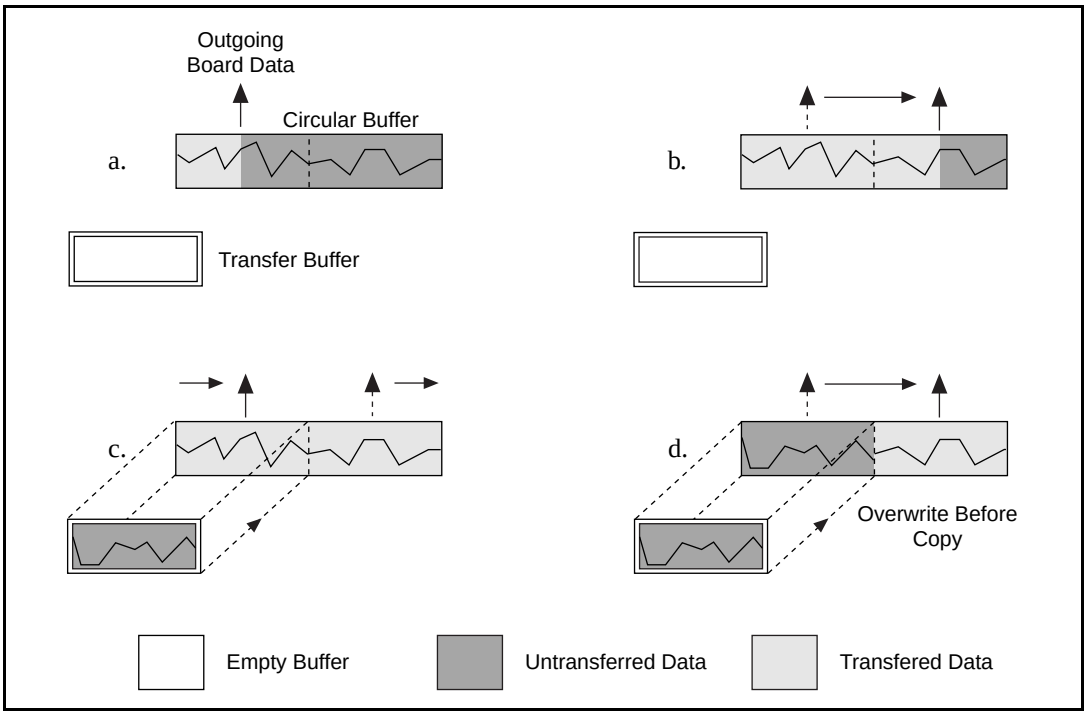


Figure 5-5. Double-Buffered Output with an Overwrite Before Copy

For this situation, NI-DAQ returns an overwrite before copy warning (**overWriteError**). This warning indicates that the device 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 device as in Figure 5-4.

The second potential problem is when an output device retrieves data that NI-DAQ is simultaneously overwriting with data from the transfer buffer. NI-DAQ returns an overwrite error (**overWriteError**) when this occurs. The situation is presented in Figure 5-6.

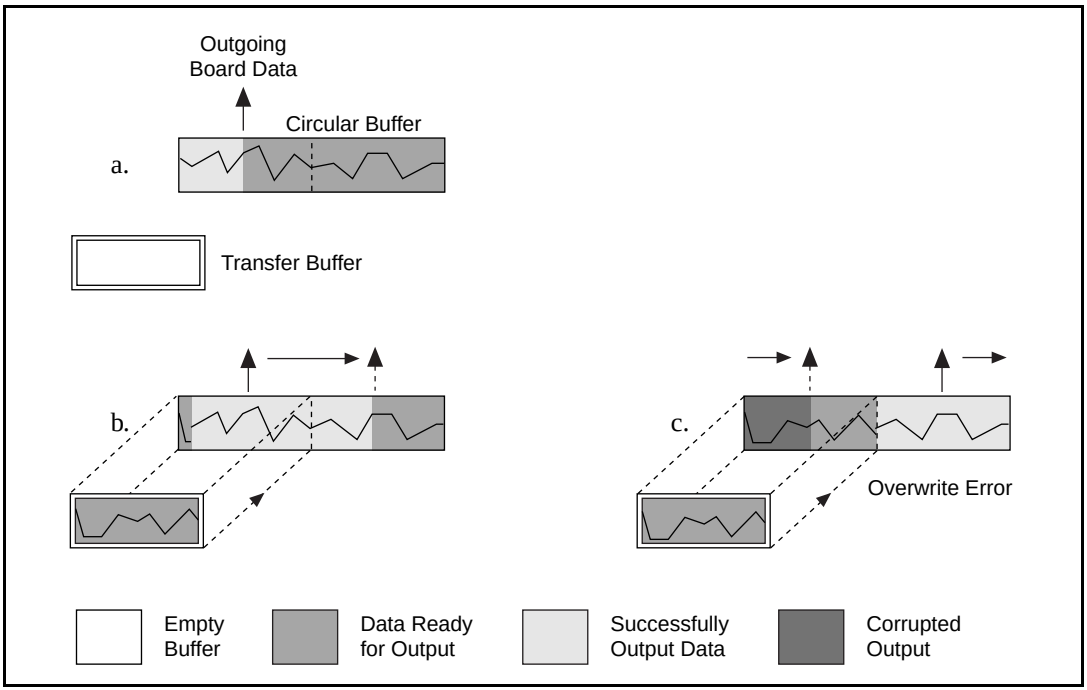


Figure 5-6. Double-Buffered Output with an Overwrite

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 device begins retrieving data from the first half (Figure 5-6c). Consequently, data output by the device might be corrupted; that is, it might 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 you can 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 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 might return an overwrite before copy warning or an overwrite error (**overWriteError**) 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 might 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 allocated to complete a synchronous function call for a device.

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.

DB_HalfReady Functions

With the `DB_HalfReady` functions, applications can avoid the delay that can occur when calling the `DB_Transfer` function. 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 might 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 device is writing or reading data. For many applications, this requirement depends on the speed and efficiency of the computer system and programming language.

Click here to comment on this document
via the National Instruments website at
http://www.natinst.com/documentation/daq/

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 Fax-on-Demand 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
Up to 14,400 baud, 8 data bits, 1 stop bit, no parity

United Kingdom: 01635 551422
Up to 9,600 baud, 8 data bits, 1 stop bit, no parity

France: 01 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.

Click here to comment on this document
via the National Instruments website at
http://www.natinst.com/documentation/daq/

Fax-on-Demand Support

Fax-on-Demand is a 24-hour information retrieval system containing a library of documents on a wide range of technical information. You can access Fax-on-Demand from a touch-tone telephone at (512) 418-1111.

E-Mail Support (currently U.S. only)

You can submit technical support questions to the applications engineering team through e-mail at the Internet address listed below. Remember to include your name, address, and phone number so we can contact you with solutions and suggestions.

`support@natinst.com`

Telephone and Fax Support

National Instruments has branch offices all over the world. Use the list below to find the technical support number for your country. If there is no National Instruments office in your country, contact the source from which you purchased your software to obtain support.

	Telephone	Fax
Australia	03 9879 5166	03 9879 6277
Austria	0662 45 79 90 0	0662 45 79 90 19
Belgium	02 757 00 20	02 757 03 11
Canada (Ontario)	905 785 0085	905 785 0086
Canada (Quebec)	514 694 8521	514 694 4399
Denmark	45 76 26 00	45 76 26 02
Finland	09 725 725 11	09 725 725 55
France	01 48 14 24 24	01 48 14 24 14
Germany	089 741 31 30	089 714 60 35
Hong Kong	2645 3186	2686 8505
Israel	03 5734815	03 5734816
Italy	02 413091	02 41309215
Japan	03 5472 2970	03 5472 2977
Korea	02 596 7456	02 596 7455
Mexico	5 520 2635	5 520 3282
Netherlands	0348 433466	0348 430673
Norway	32 84 84 00	32 84 86 00
Singapore	2265886	2265887
Spain	91 640 0085	91 640 0533
Sweden	08 730 49 70	08 730 43 70
Switzerland	056 200 51 51	056 200 51 55
Taiwan	02 377 1200	02 737 4644
U.K.	01635 523545	01635 523154

Click here to comment on this document
via the National Instruments website at
http://www.natinst.com/documentation/daq/

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, LabWindows/CVI version, Virtual Bench, or Component Works _____

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 _____

Click here to comment on this document
via the National Instruments website at
http://www.natinst.com/documentation/daq/

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 5.1*

Edition Date: July 1997

Part Number: 321644A-01

Click here to comment on this document via the National Instruments website at http://www.natinst.com/documentation/daq/

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 (___)_____ Fax (___) _____

Mail to: Technical Publications
National Instruments Corporation
6504 Bridge Point Parkway
Austin, TX 78730-5039

Fax to: Technical Publications
National Instruments Corporation
(512) 794-5678

Glossary

Prefix	Meaning	Value
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
ε	strain
Ω	ohm
°	degree
%	percent
+	plus
-	minus
±	plus or minus

A

AC	alternating current
ACK	acknowledge
A/D	analog-to-digital
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
ARB	Pertaining to arbitrary waveform generation (DAQArb 5411 devices only).
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

DDS — Direct Digital Synthesis

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 and SCXI-2400, which are hybrids.

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.

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

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

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

NC — Normally Closed

NIVDMAD — National Instruments Virtual DMA Driver

NIVISRD — National Instruments Virtual Interrupt Service Routine Driver. See the NIVISRD entry in the *Index* to locate information about the National Instruments Virtual Interrupt Service Routine Driver.

NO — normally open

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.
posttriggering	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
Remote SCXI	An SCXI configuration in which a serial port cable is connected to an SCXI-2000 chassis or an SCXI-100*X* chassis with an SCXI-2400 remote communications module. Multiple Remote SCXI units can be connected to one serial port in a PC by using RS-485. You can use either an RS-485 interface card in your PC or an RS-485 converter on the RS-232 port.
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.
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.
sequence list	(54*XX* boards only) A list of entries used in staging-based waveform generation for linking, looping, and generating multiple waveforms stored in the onboard memory.
Single-Ended (SE) Inputs	An analog input that is measured with respect to a common ground.
software trigger	A programmed event that triggers an event such as data acquisition.
S/s	Samples per second; used to express the rate at which a DAQ board samples an analog signal.
stage	(54*XX* boards only) An entry in a sequence list.

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.
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