DAQ

NI-DAQ™ User Manual for PC Compatibles

Version 6.9
Data Acquisition Software for the PC

NATIONAL INSTRUMENTS™

October 2000 Edition
Part Number 371644A-01

Worldwide Technical Support and Product Information

ni.com

National Instruments Corporate Headquarters

11500 North Mopac Expressway Austin, Texas 78759-3504 USA Tel: 512 794 0100

Worldwide Offices

Australia 03 9879 5166, Austria 0662 45 79 90 0, Belgium 02 757 00 20, Brazil 011 284 5011,
Canada (Calgary) 403 274 9391, Canada (Ontario) 905 785 0085, Canada (Québec) 514 694 8521,
China 0755 3904939, Denmark 45 76 26 00, Finland 09 725 725 11, France 01 48 14 24 24,
Germany 089 741 31 30, Greece 30 1 42 96 427, Hong Kong 2645 3186, India 91805275406,
Israel 03 6120092, Italy 02 413091, Japan 03 5472 2970, Korea 02 596 7456, Mexico (D.F.) 5 280 7625,
Mexico (Monterrey) 8 357 7695, Netherlands 0348 433466, New Zealand 09 914 0488, Norway 32 27 73 00,
Poland 0 22 528 94 06, Portugal 351 1 726 9011, Singapore 2265886, Spain 91 640 0085,
Sweden 08 587 895 00, Switzerland 056 200 51 51, Taiwan 02 2528 7227, United Kingdom 01635 523545

For further support information, see the *Technical Support Resources* appendix. To comment on the documentation, send e-mail to techpubs@ni.com

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

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

Copyright

Trademarks

WARNING REGARDING USE OF NATIONAL INSTRUMENTS PRODUCTS

(1) NATIONAL INSTRUMENTS PRODUCTS ARE NOT DESIGNED WITH COMPONENTS AND TESTING FOR A LEVEL OF RELIABILITY SUITABLE FOR USE IN OR IN CONNECTION WITH SURGICAL IMPLANTS OR AS CRITICAL COMPONENTS IN ANY LIFE SUPPORT SYSTEMS WHOSE FAILURE TO PERFORM CAN REASONABLY BE EXPECTED TO CAUSE SIGNIFICANT INJURY TO A HUMAN.

(2) IN ANY APPLICATION, INCLUDING THE ABOVE, RELIABILITY OF OPERATION OF THE SOFTWARE PRODUCTS CAN BE IMPAIRED BY ADVERSE FACTORS, INCLUDING BUT NOT LIMITED TO FLUCTUATIONS IN ELECTRICAL POWER SUPPLY, COMPUTER HARDWARE MALFUNCTIONS, COMPUTER OPERATING SYSTEM SOFTWARE FITNESS, FITNESS OF COMPILERS AND DEVELOPMENT SOFTWARE USED TO DEVELOP AN APPLICATION, INSTALLATION ERRORS, SOFTWARE AND HARDWARE COMPATIBILITY PROBLEMS, MALFUNCTIONS OR FAILURES OF ELECTRONIC MONITORING OR CONTROL DEVICES, TRANSIENT FAILURES OF ELECTRONIC SYSTEMS (HARDWARE AND/OR SOFTWARE), UNANTICIPATED USES OR MISUSES, OR ERRORS ON THE PART OF THE USER OR APPLICATIONS DESIGNER (ADVERSE FACTORS SUCH AS THESE ARE HEREAFTER COLLECTIVELY TERMED "SYSTEM FAILURES"). ANY APPLICATION WHERE A SYSTEM FAILURE WOULD CREATE A RISK OF HARM TO PROPERTY OR PERSONS (INCLUDING THE RISK OF BODILY INJURY AND DEATH) SHOULD NOT BE RELIANT SOLELY UPON ONE FORM OF ELECTRONIC SYSTEM DUE TO THE RISK OF SYSTEM FAILURE. TO AVOID DAMAGE, INJURY, OR DEATH, THE USER OR APPLICATION DESIGNER MUST TAKE REASONABLY PRUDENT STEPS TO PROTECT AGAINST SYSTEM FAILURES, INCLUDING BUT NOT LIMITED TO BACK-UP OR SHUT DOWN MECHANISMS. BECAUSE EACH END-USER SYSTEM IS CUSTOMIZED AND DIFFERS FROM NATIONAL INSTRUMENTS' TESTING PLATFORMS AND BECAUSE A USER OR APPLICATION DESIGNER MAY USE NATIONAL INSTRUMENTS PRODUCTS IN COMBINATION WITH OTHER PRODUCTS IN A MANNER NOT EVALUATED OR CONTEMPLATED BY NATIONAL INSTRUMENTS, THE USER OR APPLICATION DESIGNER IS ULTIMATELY RESPONSIBLE FOR VERIFYING AND VALIDATING THE SUITABILITY OF NATIONAL INSTRUMENTS PRODUCTS WHENEVER NATIONAL INSTRUMENTS PRODUCTS ARE INCORPORATED IN A SYSTEM OR APPLICATION, INCLUDING, WITHOUT LIMITATION, THE APPROPRIATE DESIGN, PROCESS AND SAFETY LEVEL OF SUCH SYSTEM OR APPLICATION.

Contents

About This Manual

How to Use the NI-DAQ Documentation Set ix
Conventions Used in This Manual ix
MIO and AI Device Terminology xiii

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 1-3
NI-DAQ Hardware Support 1-4
NI-DAQ Language Support 1-7
Device Configuration 1-7
Using Measurement & Automation Explorer 1-7

Chapter 2
Fundamentals of Building Windows Applications

The NI-DAQ Libraries 2-1
Creating a Windows Application Using Microsoft Visual C++ 2-2
Developing an NI-DAQ Application 2-2
Example Programs 2-2
Special Considerations 2-3
Buffer Allocation 2-3
String Passing 2-3
Parameter Passing 2-3
Creating a Windows Application Using Microsoft Visual Basic 2-4
Developing an NI-DAQ Application 2-4
Example Programs 2-5
Special Considerations 2-5
Buffer Allocation 2-5
String Passing 2-6
Parameter Passing 2-6
Creating a Windows Application Using Borland C++ 2-7
Developing an NI-DAQ Application 2-7
Example Programs 2-7
Special Considerations 2-8
Using Borland Delphi with NI-DAQ 2-8
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 Tips 3-5
NI-DAQ Events in Visual Basic for Windows 3-6
ActiveX Controls for Visual Basic 3-6
General DAQ Event 3-7
Analog Trigger Event 3-9
Analog Alarm Event 3-11
Analog Input Function Group 3-15
One-Shot Analog Input Functions 3-16
Single-Channel Analog Input Functions 3-16
Data Acquisition Functions 3-20
High-Level Data Acquisition Functions 3-20
Low-Level Data Acquisition Functions 3-21
Low-Level Double-Buffered Data Acquisition Functions 3-23
Data Acquisition Application Tips 3-24
Multirate Scanning 3-32
Analog Output Function Group 3-35
One-Shot Analog Output Functions 3-35
Analog Output Application Tips 3-36
Waveform Generation Functions 3-39
High-Level Waveform Generation Functions 3-39
Low-Level Waveform Generation Functions 3-39
Waveform Generation Application Tips 3-41
Digital I/O Function Group 3-52
DIO-24, 6025E, AT-MIO-16DE-10, DIO-96, and Lab and 1200 Device Groups 3-55
DIO-32F and 653*X* Device Groups 3-55
PCI-6115 and PCI-6120 Device Groups 3-56
Digital I/O Functions 3-57
Group Digital I/O Functions 3-58
Double-Buffered Digital I/O Functions 3-59
Digital Change Notification Functions 3-60
Digital Filtering Function 3-60
Digital Change Notification Applications with 652*X* Devices 3-60
Digital Change Detection Applications with 653*X* Devices 3-61

Digital I/O Application Tips....3-62
Handshaking Versus No-Handshaking Digital I/O....3-63
Digital Port I/O Applications....3-63
Digital Line I/O Applications....3-65
Digital Group I/O Applications....3-67
Digital Group Block I/O Applications....3-69
Digital Double-Buffered Group Block I/O Applications....3-71
Pattern Generation I/O with the DIO-32F, 653*X*, PCI-6115, and PCI-6120 Devices....3-74
Double-Buffered I/O....3-75
Counter/Timer Function Group....3-76
Counter/Timer Functions....3-77
Counter/Timer Operation for the CTR Functions....3-78
Programmable Frequency Output Operation....3-81
Counter/Timer Application Tips....3-82
Interval Counter/Timer Functions....3-91
Interval Counter/Timer Operation for the ICTR Functions....3-92
Interval Counter/Timer Application Tips....3-92
General-Purpose Counter/Timer Functions....3-93
General-Purpose Counter/Timer Application Tips....3-94
Clocks or Time Counters....3-95
Clock Resolution....3-95
Clock Synchronization....3-96
Clock Accuracy....3-98
Example Clock in a Measurement System....3-98
Sample Use Cases....3-99
RTSI Bus Trigger Functions....3-102
RTSI Bus....3-102
E Series, DSA, 660*X*, and 671*X* RTSI Connections....3-103
AT-AO-6/10 RTSI Connections....3-104
DIO-32F RTSI Connections....3-104
653*X* RTSI Connections....3-105
RTSI Bus Application Tips....3-106
SCXI Functions....3-107
SCXI Application Tips....3-112
Building Analog Input Applications in Multiplexed Mode....3-113
Building Analog Input Applications in Parallel Mode....3-120
SCXI Data Acquisition Rates....3-124
Analog Output Applications....3-126
Digital Applications....3-127

Chapter 4
NI-DAQ Double Buffering

Overview 4-1
Single-Buffered Versus Double-Buffered Data 4-1
Double-Buffered Input Operations 4-2
Potential Setbacks 4-4
Double-Buffered Output Operations 4-6
Potential Setbacks 4-7
Double-Buffered Functions 4-9
Double Buffer Configuration Functions 4-10
Double Buffer Transfer Functions 4-10
Double Buffer HalfReady Functions 4-11
Conclusion 4-12

Chapter 5
Transducer Conversion Functions

Function Descriptions 5-2
RTD_Convert and RTD_Buf_Convert 5-2
Parameter Discussion 5-2
Using This Function 5-3
Strain_Convert and Strain_Buf_Convert 5-4
Parameter Discussion 5-4
Using This Function 5-5
Thermistor_Convert and Thermistor_Buf_Convert 5-7
Parameter Discussion 5-7
Using This Function 5-8
Thermocouple_Convert and Thermocouple_Buf_Convert 5-9
Parameter Discussion 5-9
Using This Function 5-10

Appendix A
Technical Support Resources

Glossary

Index

About This Manual

The *NI-DAQ User Manual for PC Compatibles* is for users of the NI-DAQ software for *PC* compatibles version 6.9. NI-DAQ software is a powerful application programming interface (*API*) between your data acquisition (*DAQ*) application and the National Instruments DAQ *device*s. Source code for several example applications is included in this manual.

How to Use the NI-DAQ Documentation Set

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 other National Instruments application software.

When you are familiar with the material in this manual, you can begin to use the *NI-DAQ Function Reference Online Help file,* `Nidaqpc.hlp`, the Windows help file that contains detailed descriptions of the NI-DAQ functions. Other documentation includes the *DAQ Hardware Overview Guide*, and the DAQ provider help contained in Measurement & Automation Explorer.

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

Conventions Used in This Manual

The following conventions are used in this manual.

»	The » symbol leads you through nested menu items and dialog box options to a final action. The sequence **File»Page Setup»Options** directs you to pull down the **File** menu, select the **Page Setup** item, and select **Options** from the last dialog box.
	This icon denotes a note, which alerts you to important information.
1102/B/C modules	Refers to the *SCXI*-1102, SCXI-1102B, and SCXI-1102C modules and the *VXI*-SC-1102, VXI-SC-1102B, and VXI-SC-1102C submodules.
12-*bit* device	These *MIO* and *AI* devices are listed in Table 1.

1200 and 1200AI device	Refers to the DAQCard-1200, DAQPad-1200, Lab-PC-1200, Lab-PC-1200AI, PCI-1200, and SCXI-1200.
1394	Refers to a high-speed external *bus* that implements the *IEEE* 1394 serial bus protocol.
16-bit device	These MIO and AI devices are listed in Table 1.
44*XX* device	Refers to the NI 4451 for PCI, NI 4452 for PCI, NI 4454 for PCI, and NI 4472 for PXI/CompactPCI.
45*XX* device	Refers to the NI 4551 for PCI and NI 4552 for PCI.
516 device	Refers to the DAQCard-516 and PC-516.
6025E device	Refers to the PCI-6025E and *PXI*-6025E.
6052E device	Refers to the PCI-6052E, PXI-6052E, DAQPad-6052E for1394, and DAQPad-6052E for USB.
6053E device	Refers to the PCI-6053E and PXI-6053E.
61*XX* device	Refers to the PCI-6110, PCI-6111, PCI-6115, PXI-6115, PCI-6120, and PXI-6120.
622*X* device	Refers to the NI-6222 for PCI, NI-6222 for PXI, and NI-6224 for Ethernet.
652*X* device	Refers to the PCI-6527 and PXI-6527.
653*X* device	Refers to the AT-DIO-32HS, PCI-DIO-32HS, DAQCard-6533, PXI-6533, PCI-6534, and PXI-6534.
660*X* device	Refers to the DAQCard-6601, PCI-6601, PCI-6602, PXI-6602, PCI-6608, and PXI-6608.
6602 device	Refers to the PCI-6602 and PXI-6602.
671*X* device	Refers to the DAQCard-6715, PCI-6711, PXI-6711, PCI-6713, and PXI-6713.
AI device	These analog input devices are listed in Table 1.
bold	Bold text denotes items that you must select or click on in the software, such as menu items and dialog box options. Bold text also denotes parameter names and function prototypes.
DAQCard-500/700	Refers to the DAQCard-500 and DAQCard-700.

DIO device	Refers to any DIO-24, DIO-32, DIO-6533, or DIO-96.
DIO-24	Refers to the PC-DIO-24, PC-DIO-24PnP, DAQCard-DIO-24, PCI-6503.
DIO-32F	Refers to the AT-DIO-32F.
DIO-96	Refers to the PC-DIO-96, PC-DIO-96PnP, PCI-DIO-96, DAQPad-6507, DAQPad-6508, and PXI-6508.
DSA device	Refers to the NI 4451 for PCI, NI 4452 for PCI, NI 4454 for PCI, NI 4551 for PCI, NI 4552 for PCI, and NI 4472 for PXI/CompactPCI dynamic signal acquisition devices.
E Series device	These are MIO and AI devices. Refer to Table 1 for a complete list of these devices.
italic	Italic text denotes variables, emphasis, a cross reference, or an introduction to a key concept. This font also denotes text that is a placeholder for a word or value that you must supply.
Lab and 1200 analog output device	Refers to the DAQCard-1200, DAQPad-1200, Lab-PC+, Lab-PC-1200, PCI-1200, and SCXI-1200.
Lab and 1200 device	Refers to the DAQCard-1200, DAQPad-1200, Lab-PC+, Lab-PC-1200, Lab-PC-1200AI, 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. See Table 1 for a list of these devices.
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-64E-3, 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 enter from the keyboard, sections of code, programming examples, and syntax examples. This font is also used for the proper names of disk drives, paths, directories, programs, subprograms, subroutines, device names, functions, operations, properties, methods, variables, filenames and extensions, and code excerpts.
`monospace bold`	Bold text in this font denotes the messages and responses that the computer automatically prints to the screen. This font also emphasizes lines of code that are different from the other examples.

monospace italic	Italic text in this font denotes text that is a placeholder for a word or value that you must supply.
NI-DAQ	Refers to the NI-DAQ software for PC compatibles, unless otherwise noted.
NI-TIO based device	Refers to the NI 4551, NI 4552, DAQCard-6601, PCI-6601, PCI-6602, PXI-6602, PCI-6608, PXI-6608.
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.
PXI	Refers to PCI eXtensions for Instrumentation, derived from the CompactPCI standard.
remote SCXI	Refers to an SCXI configuration where either an SCXI-2000 chassis or an SCXI-2400 remote communications module is connected to the PC serial *port*.
SCXI-1102/B/C	SCXI-1102/B/C refers to the SCXI-1102, SCXI-1102B, and SCXI-1102C.
SCXI-1120/D	SCXI-1120/D refers to the SCXI-1120 and SCXI-1120D.
SCXI-1104/C	SCXI-1104/C refers to the SCXI-1104 and SCXI-1104C.
SCXI analog input module	Refers to the SCXI-1100, SCXI-1101, SCXI-1102, SCXI-1104,SCXI-1104C, SCXI-1112, SCXI-1120, SCXI-1120D, SCXI-1121, SCXI-1122, SCXI-1125, SCXI-1140, SCXI-1141, SCXI-1142, SCXI-1143, and SCXI-1520.
SCXI analog output module	Refers to the SCXI-1124 module.
SCXI chassis	Refers to the SCXI-1000, SCXI-1000*DC*, SCXI-1001, and SCXI-2000.
SCXI digital module	Refers to the SCXI-1160, SCXI-1161, SCXI-1162, SCXI-1162HV, SCXI-1163, and SCXI-1163R.
SCXI switch module	Refers to the SCXI-1190, SCXI-1191, SCXI-1127, and SCXI-1128.
simultaneous sampling device	Refers to the PCI-6110, PCI-6111, PCI-6115, PXI-6115, PCI-6120, PXI-6120, NI 4451 for PCI, NI 4452 for PCI, NI 4454 for PCI, NI 4551 for PCI, NI 4552 for PCI, and NI 4472 for PXI/CompactPCI.

track-and-hold module	Refers to the SCXI-1140, SCXI-1520, SCXI-1530, and SCXI-1531.
VXI-MIO device	Refers to the VXI-MIO-64E-1 and VXI-MIO-64XE-10.
VXI-SC-1102/B/C	Refers to the VXI-SC-1102, VXI-SC-1102B, and VXI-SC-1102C.

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, and the functionality. These devices are also collectively known as E Series devices. 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
AT-AI-16XE-10	16	16-bit	AT	AI
AT-MIO-16DE-10	16	12-bit	AT	MIO
AT-MIO-16E-1	16	12-bit	AT	MIO
AT-MIO-16E-2	16	12-bit	AT	MIO
AT-MIO-16E-10	16	12-bit	AT	MIO
AT-MIO-16F-5	16	12-bit	AT	MIO
AT-MIO-16XE-10	16	16-bit	AT	MIO
AT-MIO-16XE-50	16	16-bit	AT	MIO
AT-MIO-64E-3	64	12-bit	AT	MIO
DAQCard-6023E	16	12-bit	PCMCIA	AI
DAQCard-6024E	16	12-bit	PCMCIA	MIO
DAQCard-6062E	16	12-bit	PCMCIA	MIO
DAQCard-AI-16E-4	16	12-bit	PCMCIA	AI
DAQCard-AI-16XE-50	16	16-bit	PCMCIA	AI
DAQPad-MIO-16XE-50	16	16-bit	Parallel Port	MIO
DAQPad-6020E	16	12-bit	USB	MIO
DAQPad-6052E for 1394	16	16-bit	1394	MIO
DAQPad-6052E for USB	16	16-bit	USB	MIO

Table 1. MIO and AI Device Classifications (Continued)

Device	Number of SE Channels	Bit	Type	Functionality
DAQPad-6070E	16	12-bit	1394	MIO
NEC-AI-16E-4	16	12-bit	NEC	AI
NEC-AI-16XE-50	16	16-bit	NEC	AI
NEC-MIO-16E-4	16	12-bit	NEC	MIO
NEC-MIO-16XE-50	16	16-bit AI,	NEC	MIO
NI 6222 for PCI	24	16-bit AI, 12-bit AO	PCI	MIO
NI 6222 for PXI	24	16-bit AI, 12-bit AO	PXI	MIO
NI 6224 for Ethernet	56	16-bit AI, 12-bit AO	Ethernet	MIO
PCI-6023E	16	12-bit	PCI	AI
PCI-6024E	16	12-bit	PCI	MIO
PCI-6025E	16	12-bit	PCI	MIO
PCI-6031E (MIO-64XE-10)	64	16-bit	PCI	MIO
PCI-6032E (AI-16XE-10)	16	16-bit	PCI	AI
PCI-6033E (AI-64XE-10)	64	16-bit	PCI	AI
PCI-6034E	16	16-bit	PCI	AI
PCI-6035E	16	16-bit AI, 12-bit AO	PCI	MIO
PCI-6052E	16	16-bit	PCI	MIO
PCI-6053E	64	16-bit	PCI	MIO
PCI-6071E (MIO-64E-1)	64	12-bit	PCI	MIO
PCI-6110	4, DIFF only	12-bit AI, 16-bit AO	PCI	MIO
PCI-6111	2, DIFF only	12-bit AI, 16-bit AO	PCI	MIO
PCI-6115	4, DIFF only	12-bit	PCI	MIO
PCI-6120	4, DIFF only	16-bit	PCI	MIO
PCI-MIO-16E-1	16	12-bit	PCI	MIO
PCI-MIO-16E-4	16	12-bit	PCI	MIO
PCI-MIO-16XE-10	16	16-bit	PCI	MIO
PCI-MIO-16XE-50	16	16-bit	PCI	MIO
PXI-6011E	16	16-bit	PXI	MIO

Table 1. MIO and AI Device Classifications (Continued)

Device	Number of SE Channels	Bit	Type	Functionality
PXI-6023E	16	12-bit	PXI	AI
PXI-6024E	16	12-bit	PXI	MIO
PXI-6025E	16	12-bit	PXI	MIO
PXI-6030E	16	16-bit	PXI	MIO
PXI-6031E	64	16-bit	PXI	MIO
PXI-6034E	16	16-bit	PXI	AI
PXI-6035E	16	16-bit AI, 12-bit AO	PXI	MIO
PXI-6040E	16	12-bit	PXI	MIO
PXI-6052E	16	16-bit	PXI	MIO
PXI-6053E	64	16-bit	PXI	MIO
PXI-6070E	16	12-bit	PXI	MIO
PXI-6115	4, DIFF only	12-bit	PXI	MIO
PXI-6120	4, DIFF only	16-bit	PXI	MIO
VXI-MIO-64E-1	64	12-bit	VXI	MIO
VXI-MIO-64XE-10	64	16-bit	VXI	MIO

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 the DAQ device registers 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 multiboard 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 are:

- 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
- When a rising or falling edge occurred on a timing I/O line

How to Set Up Your DAQ System

After you have installed your software and hardware and configured your hardware, see Figure 1-1 to begin using NI-DAQ in your application programs.

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

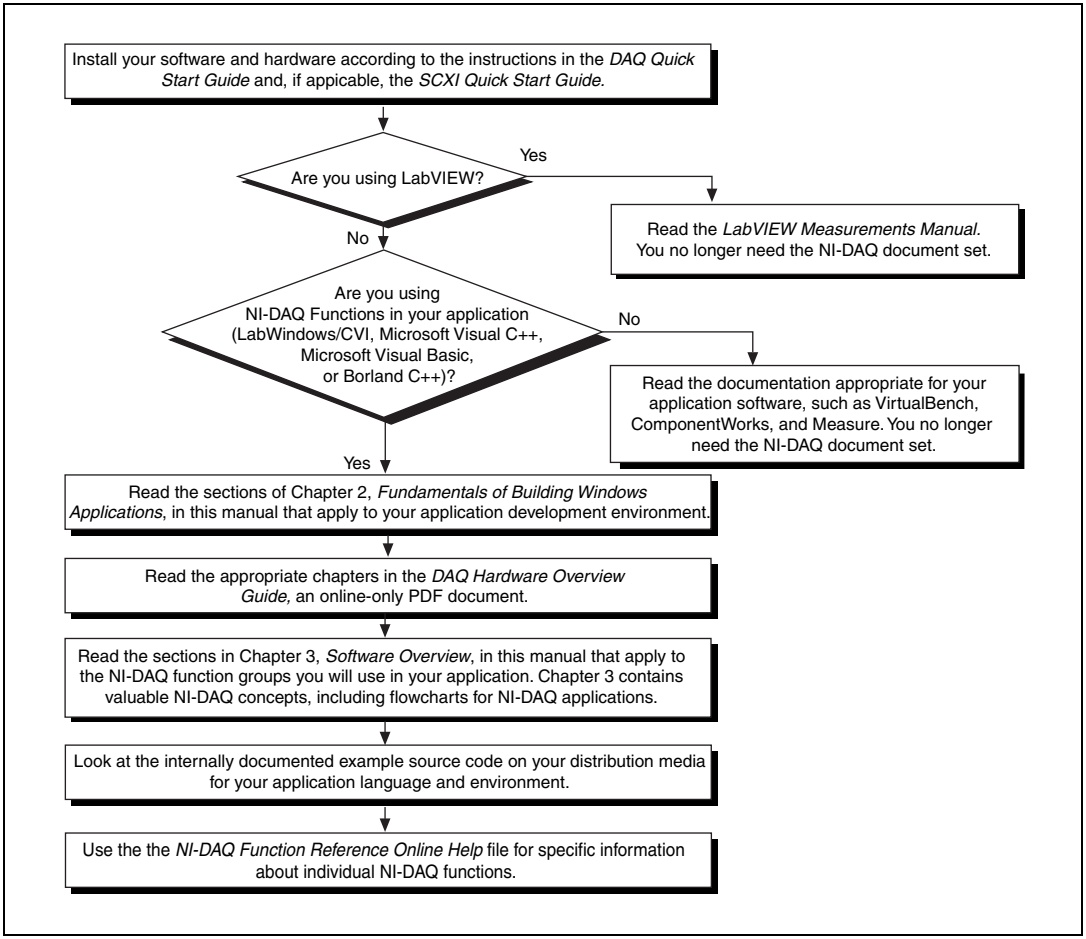


Figure 1-1. How to Set up Your DAQ System

NI-DAQ Overview

NI-DAQ is a library of routines that work with National Instruments DAQ devices. NI-DAQ helps you overcome 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 your hardware, consult Tables 1-1 through 1-5.

Table 1-1. NI-DAQ Version 6.9 Plug-in Device Support for AT, PC, and NEC Buses

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

Table 1-2. NI-DAQ Version 6.9 Plug-in Device Support for PCI Buses

PCI		
NI 4451 for PCI	PCI-DIO-96	PCI-6115
NI 4452 for PCI	PCI-1200	PCI-6120
NI 4454 for PCI	PCI-6023E	PCI-6503
NI 4551 for PCI	PCI-6024E	PCI-6527
NI 4552 for PCI	PCI-6025E	PCI-6534
NI 6222 for PCI	PCI-6031E (MIO-64XE-10)	PCI-6602
PCI-MIO-16E-1	PCI-6032E (AI-16XE-10)	PCI-6608
PCI-MIO-16E-4	PCI-6033E (AI-64XE-10)	PCI-6703
PCI-MIO-16XE-10	PCI-6052E	PCI-6704
PCI-6034E	PCI-6071E (MIO-64E-1)	PCI-6711
PCI-6035E	PCI-6110	PCI-6713
PCI-MIO-16XE-50	PCI-6111	PCI-4452
PCI-DIO-32HS		

Table 1-3. NI-DAQ Version 6.9 Plug-in Device Support for PXI Buses

PXI		
NI 4472 for PXI/CompactPCI	PXI-6035E	PXI-6527
NI 6222 for PXI	PXI-6040E	PXI-6533
PXI-6023E	PXI-6052E	PXI-6534
PXI-6024E	PXI-6070E	PXI-6602
PXI-6025E	PXI-6071E	PXI-6608
PXI-6030E	PXI-6115	PXI-6703
PXI-6031E	PXI-6120	PXI-6704
PXI-6034E	PXI-6508	

Table 1-4. NI-DAQ Version 6.9 Plug-in Device Support for PC Card, CardBus, and VXI Buses

PC Card		CardBus	VXI
DAQCard-500	DAQCard-6533	DAQCard-6601	VXI-MIO-64E-1
DAQCard-516	DAQCard-6715	DAQCard-6533	VXI-MIO-64XE-10
DAQCard-700	DAQCard-AI-16E-4		VXI-DIO-128
DAQCard-1200	DAQCard-AI-16XE-50		VXI-AO-48XDC
DAQCard-6023E	DAQCard-AO-2DC		
DAQCard-6024E	DAQCard-DIO-24		
DAQCard-6062E			

Table 1-5. NI-DAQ Version 6.9 External Device Support for SCXI and USB Devices

SCXI			USB[2]
PXI-1010	SCXI-1120D	SCXI-1200[1]	DAQPad-6507
PXI-1011	SCXI-1121	SCXI-1520	DAQPad-6508
SCXI-1000	SCXI-1122	SCXI-1530	DAQPad-6020E
SCXI-1000DC	SCXI-1124	SCXI-1531	DAQPad-6052E for USB
SCXI-1001	SCXI-1125	SCXI-1540	
SCXI-1010	SCXI-1126	SCXI-2000	
SCXI-1011	SCXI-1140	SCXI-2400	
SCXI-1100	SCXI-1141	SCXI-1122	
SCXI-1101	SCXI-1142	VXI-SC-1000	
SCXI-1102	SCXI-1143	VXI-SC-1102	
SCXI-1102B	SCXI-1160	VXI-SC-1102B	
SCXI-1102C	SCXI-1161	VXI-SC-1102C	
SCXI-1104	SCXI-1162	VXI-SC-1150	
SCXI-1104C	SCXI-1162HV		
SCXI-1112	SCXI-1163		
SCXI-1120	SCXI-1163R		

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

Table 1-6. NI-DAQ Version 6.9 External Device Support for Parallel Port, 1394, and Other Devices

Parallel Port[2]	1394	Other Devices
DAQPad-1200	DAQPad-6052E for 1394	AMUX-64T
DAQPad-MIO-16XE-50	DAQPad-6070E	SC-2040
		SC-2042-RTD
		SC-2043-SG
		SC-2345
		NI 6224 for Ethernet

Throughout this manual, many of the devices are grouped into categories that are similar in functionality. The categories are often used in the text to avoid long lists of specific devices. The *Conventions Used in This Manual* section of *About This Manual* lists the devices in each functional type. Any device not included in a category will be referred to 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 2000/NT/98/95:

- Microsoft Visual C++ 4.*X*, 5.0, or 6.0
- Visual Basic 4.0 (32-bit), 5.0, or 6.0
- Borland C++ 5.*X*

NI-DAQ also provides an NI-DAQ function prototype file for use with Borland Delphi 2 (32-bit), 3, and 4.

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 NI-DAQ Readme File (**Start»Programs»National Instruments DAQ»NI-DAQ Readme 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, PC cards (PCMCIA), or external devices you connect to the parallel port of your computer. NI-DAQ needs the device configuration information to program your hardware properly.

Using Measurement & Automation Explorer

Measurement & Automation Explorer is a Windows-based application that you use to configure and view National Instruments DAQ device settings under Windows 2000/NT/98/95.

Note To use Measurement & Automation Explorer, quit any applications that are performing DAQ operations.

Double-click the **Measurement & Automation** icon on your desktop to run Measurement & Automation Explorer. Refer to the Measurement & Automation Explorer online help for more information and detailed instructions.

2

Fundamentals of Building Windows Applications

This chapter describes the fundamentals of creating NI-DAQ applications in Windows 2000/NT/98/95.

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 the 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 appropriate import library. See the following sections for language-specific details.

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 development tools that NI-DAQ works with, you must create your own function prototype file.

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, and that you are familiar with the IDE.

Developing an NI-DAQ Application

To develop an NI-DAQ application, follow these general steps:

1. Open an existing or new Visual C++ project 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. Make sure you include the NI-DAQ header file, `nidaq.h`, as such in your source code files:

   ```
   #include "nidaq.h"
   ```

   You may also want to include `nidaqcns.h` and `nidaqerr.h`. Optionally, you can include other files (for example, `.rc, .def`) that you have created for graphical user interface (GUI) applications.
3. Specify the directory which contains the NI-DAQ header files under the **preprocessor»include directory** settings in your compiler. (For Visual C++ 4.*X*, this is under **Build»Settings»C/C++**. For Visual C++ 5.0/6.0, this is under **Project»Settings»C/C++**.) The NI-DAQ header files are located in the `.\Include` directory under your NI-DAQ directory.
4. Add the NI-DAQ import library `nidaq32.lib` to the project. The NI-DAQ import library files are located in the `.\Lib` directory under your NI-DAQ directory.
5. Build your application.

Example Programs

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

To load an example program, use one of the generic makefiles with the `.mak` extension.

To load an example project with Visual C++ 4.*X* or later, select the menu option **File»Open Project Workspace**, and select **List Files of Type** to be **Makefiles**. Then select the `.mak` file of your choice.

Refer to the **NI-DAQ Examples Help (Start»Programs»National Instruments DAQ»NI-DAQ Examples Help**) for additional information regarding NI-DAQ examples.

Special Considerations

Buffer Allocation

To allocate memory, you can use the Windows API function `GlobalAlloc()`. After allocation, lock memory with `GlobalLock()` to use a buffer of memory. You can use the memory handle returned by `GlobalLock()` in place of the **buffer** parameter in NI-DAQ API functions that accept buffers (`Align_DMA_Buffer`, `DAQ_DB_Transfer`, `DAQ_Monitor`, `DAQ_Op`, `DAQ_Start`, `DIG_Block_In`, `DIG_Block_Out`, `DIG_DB_Transfer`, `GPCTR_Config_Buffer`, `GPCTR_Read_Buffer`, `Lab_ISCAN_Op`, `Lab_ISCAN_Start`, `SCAN_Op`, `SCAN_Start`, `SCAN_Sequence_Demux`, `WFM_DB_Transfer`, `WFM_Load`, `WFM_Op`). After using the memory, unlock memory with `GlobalUnlock()` and free it with `GlobalFree()`.

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

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 variables when you need to pass by address.

Creating a Windows Application Using Microsoft Visual Basic

This section assumes that you will be using the Microsoft Visual Basic IDE to manage your code development, and that you are familiar with the IDE.

Developing an NI-DAQ Application

To develop an NI-DAQ application, follow these general steps:

1. Open an existing or new Visual Basic project to manage your application code.
2. Create files of type `.frm` (form definition and event handling code), `.bas` (Visual Basic generic code module), or `.cls` (Visual Basic class module) and add them to the project.
3. Include the NI-DAQ include file for Visual Basic, `nidaq32.bas`, into your project. You may also want to include `nidaqcns.inc` and `nidaqerr.inc`. The NI-DAQ include files for Visual Basic are located in the `.\Include` directory under your NI-DAQ directory. For Visual Basic 5.0/6.0, you can select the **Project»Add Module** menu option, click on the **Existing** tab, then select the module of your choice.

   Alternatively, you can add a reference to the National Instruments Data Acquisition Type Library, which is part of the NI-DAQ DLL. In Visual Basic 5.0/6.0, select the **Project»References** menu option, and check National Instruments Data Acquisition Library. If you do not see it listed there, click on the **Browse** button and locate `nidaq32.dll` in your `\Windows\system` or `\Windows\system32` directory.
4. Run your application by clicking the **Run** button.

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 Language Considerations* topic in the *NI-DAQ Function Reference Online Help* file 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 `.\Examples\VBasic` directory under your NI-DAQ directory.

To load an example program, use one of the Visual Basic project files with the `.vbp` extension. These are Visual Basic 4.0 projects, which you can open only with Visual Basic 4.0 or later.

To load an example project with Visual Basic 4.0 or later, select the menu option **File»Open Project**, then select the `.vbp` file of your choice.

Refer to the **NI-DAQ Examples Help (Start»Programs»National Instruments DAQ»NI-DAQ Examples Help**) for additional information regarding NI-DAQ examples.

Special Considerations

Buffer Allocation

Visual Basic 4.0 is quite restrictive when allocating memory. You allocate memory by declaring an array of the 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) long integers, or 8,191 doubles).

To break the 64 KB buffer size barrier, you can use the Windows API functions `GlobalAlloc()` to allocate buffers larger than 64 KB. After allocation, you must lock memory with `GlobalLock()` to use a buffer of memory. You can use the memory handle returned by `GlobalLock()` in place of the buffer parameter in NI-DAQ API functions that accept buffers (`Align_DMA_Buffer`, `DAQ_DB_Transfer`, `DAQ_Monitor`, `DAQ_Op`, `DAQ_Start`, `DIG_Block_In`, `DIG_Block_Out`, `DIG_DB_Transfer`, `GPCTR_Config_Buffer`, `GPCTR_Read_Buffer`, `Lab_ISCAN_Op`, `Lab_ISCAN_Start`, `SCAN_Op`, `SCAN_Start`, `SCAN_Sequence_Demux`, `WFM_DB_Transfer`, `WFM_Load`, `WFM_Op`). The NI-DAQ header file declares the buffer parameter "As Any." After using the memory, you must unlock memory with `GlobalUnlock()` and free it with `GlobalFree()`.

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

The following paragraph illustrates declarations of functions.

For Visual Basic 4.0 or later, 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.

Creating a Windows Application Using Borland C++

This section assumes that you will be using the Borland C++ IDE to manage your code development, and that you are familiar with the IDE.

Developing an NI-DAQ Application

To develop an NI-DAQ application, follow these general steps:

1. Open an existing or new Borland C++ project 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.
   - Make sure you include the NI-DAQ header file, `nidaq.h`, as such in your source code files:

     ```
     #include "nidaq.h"
     ```

   - You may also want to include `nidaqcns.h` and `nidaqerr.h`.
   - Optionally, you can include other files (for example, `.rc`, `.def`) for GUI applications.
3. Specify the directory that contains the NI-DAQ header and import library files under the **source directories** (**Include, Libary**) settings of your compiler. For Borland C++ 5.0, this directory is under **Options»Projects»Directories**. The NI-DAQ header files are located in the `.\Include` directory under your NI-DAQ directory, and the import library files are located in the `.\Lib` directory under your NI-DAQ directory.
4. Add the NI-DAQ import library, `nidaq32.lib`, to the project.
5. Build your application.

Example Programs

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

To build an example program, run one of the batch files with the `.bat` extension from a DOS prompt. You will have to modify the batch file to set one of the environmental variables to point to your Borland C++ IDE directory. If you open one of the batch files with a text editor, you will see the following line:

```
set BorlandDir=e:\apps\bc5
```

Change the right hand side of the equal sign to indicate your Borland C++ IDE directory. For help on the usage of the batch file, type `<batchfile.bat> /?` from a DOS prompt, where `<batchfile.bat>` is the file name of batch file you want to run (for example, `AIonePoint.bat`).

To create your own example project with Borland C++ 5.0 or later using the provided example files, follow the steps mentioned above in **Developing an NI-DAQ Application**. In place of `nidaq.h`, make sure you include `nidaqex.h`. Also, make sure you include the import library `nidex32b.lib` into your project in addition to `nidaq32b.lib`.

Refer to the **NI-DAQ Examples Help** (**Start»Programs»National Instruments DAQ»NI-DAQ Examples Help**) for additional information regarding NI-DAQ examples.

Special Considerations

Refer to *Special Considerations* in the *Creating a Windows Application Using Microsoft Visual C++* section.

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 written with the NI-DAQ API for Borland Delphi. For examples on NI-DAQ function flow, refer to the examples of other languages and the flowchart in Chapter 3, *Software Overview*. Refer to the note at the end of the *NI-DAQ Examples* section of this chapter for information on examples using ComponentWorks ActiveX controls.

NI-DAQ Examples

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

- LabWindows/CVI 5.0.*x*
- Microsoft Visual C++ 2.*x* (32-bit) or later
- Microsoft Visual Basic 4.0 (32-bit) or later
- Borland C++ 5.0

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 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 can simply change the assignment.

The examples correspond to the function flowcharts that you will see in Chapter 3, *Software Overview*. If a task and a flowchart 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++, where `.bat` files are included to help build the executable)
- 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 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
`.\CALIB`	Calibration examples

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

- LabWindows/CVI:

  `.prj` (project file), `.c` (source file)

- Visual C++:

  `.mak` (generic make file), `.c` (source file)

- Visual Basic:

  `.vbp` (project file, for Visual Basic 4.0 [32-bit] or later),
  `.frm` (form module)

- Borland C++:

  `.bat` (Batchfile), `.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»National Instruments DAQ»NI-DAQ Examples Help**. You will have this file only if you installed support for LabWindows/CVI, Visual C++, Visual Basic, or Borland C++ examples.

In addition to examples using the NI-DAQ API, you can install and use examples using the ComponentWorks DAQ controls.

Run the NI-DAQ Setup Utility (**Start»Programs»National Instruments DAQ»NI-DAQ Setup**) and choose either of the following:

- Microsoft Visual Basic (under **Details**, select **ComponentWorks ActiveX Controls** and **ComponentWorks DAQ Visual Basic Examples**)
- Borland Delphi (under **Details**, select **ComponentWorks ActiveX Controls** and **ComponentWorks DAQ Borland Delphi Examples**)

These examples will be installed in the `.\Examples\Visual Basic` and `.\Examples\Borland Delphi` directories, respectively, under your NI-DAQ directory.

3

Software Overview

This chapter describes the function classes in NI-DAQ and briefly describes each function.

NI-DAQ functions are grouped according to the following classes:

- Initialization and general-configuration
- Software-calibration and device-specific
- Event message
- 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
  - Change Notification
  - Filtering
- 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.

Refer to the *NI-DAQ Function Reference Online Help* file to determine which functions your device supports.

`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.
`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.

Refer to the *NI-DAQ Function Reference Online Help* file to determine which functions your device supports.

`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 1200/AI devices 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_TIO`	Use the function to calibrate the crystal oscillator on your timing I/O 660*X* device.
`Calibrate_DSA`	Use this function to calibrate your DSA device.
`Calibrate_E_Series`	Use this function to calibrate your E Series device or 671*X* 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 LPM device 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_Config`	Turns dithering on and off. For the MIO-64, this function also lets you specify whether to use AMUX-64T channels or onboard channels.
`SCXI_Calibrate`	Performs a self-calibration (or internal calibration) for certain SCXI modules.
`Select_Signal`	Selects the source and polarity of certain signals used by the E Series and DSA devices. You typically need to use this function to externally control timing, to use the RTSI bus, or to configure one of the I/O connector PFI pins.

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 Tips

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.

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 occur, 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.

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`.

To add or remove a message, first clear your data acquisition process. Then, call one of the three event message configuration functions.

NI-DAQ Events in Visual Basic for Windows

ActiveX Controls for Visual Basic

Unlike standard control-flow programming languages, event occurrences drive 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 run when you click the **Run** button, enter code in the `Run_Click()` procedure. When a program starts executing, Visual Basic looks for events related to controls and calls control procedures as necessary. You do not write an event loop.

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

- General Data Acquisition Event (`daqevent.ocx`)

- Analog Trigger Event (`atrigev.ocx`)

- Analog Alarm Event (`alarmev.ocx`)

The NI-DAQ installer places all of these ActiveX controls in the `NIDAQ` subdirectory of your Windows 2000/NT/98/95 directory under the file names shown.

These three ActiveX 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 ActiveX controls shown previously in this section, your Visual Basic application can receive NI-DAQ messages.

Note You can use the OCXs in Visual Basic, version 4.0 (32-bit) or later.

General DAQ Event

You use the General DAQ Event 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 Online Help* file.
`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

Table 3-1. General DAQ Event Control Properties (Continued)

Property	Allowed Property Values
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 description for the `Config_DAQ_Event_Message` function in the *NI-DAQ Function Reference Online Help* file.

Set each of these properties as follows:

`GeneralDAQEventn.property name = property value`

For example, to set the `ChanStr` property to Analog Input channel 0 for GeneralDAQEvent 1:

```
GeneralDAQEvent1.ChanStr = "AI0"
```

Set up your program flow like this:

1. Set the properties of the General DAQ Event control. Then, configure the acquisition or generation operations using the appropriate NI-DAQ functions.
2. Set the `Enabled` property of the General DAQ Event control to 1 (True).
3. Invoke the `GeneralDAQEventn.Refresh` method to set the DAQ Event in the NI-DAQ driver. Each subsequent use of `GeneralDAQEventn.Refresh` deletes the old DAQ Event and sets a new one with the current set of properties.

4. Start an asynchronous data acquisition or generation operation.
5. When the selected event occurs, the `GeneralDAQEventn_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 `GeneralDAQEventn_Fire` procedure is prototyped as follows:

`Sub GeneralDAQEventn_Fire` (DoneFlag As Integer, Scans As Long)

The parameter **DoneFlag** equals 1 if the acquisition was over 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 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

Use the Analog Trigger Event 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 Online Help* file
`Level`	Single (voltage)
`WindowSize`	Single (voltage)
`Slope`	0—Positive (default) 1—Negative
`TrigSkipCount`	Long
`PreTrigScans`	Long
`PostTrigScans`	Long

Table 3-2. Analog Trigger Event Control Properties (Continued)

Property	Allowed Property Values
`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 Online Help* file.

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"
```

Set up your program flow like this:

1. Set the properties of the Analog Trigger Event control. Next, 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 `AnalogTriggerEventn_Fire` procedure is prototyped as follows:

`Sub AnalogTriggerEventn_Fire` (DoneFlag As Integer, Scans As Long)

The parameter **DoneFlag** equals 1 if the acquisition was over 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

Use the Analog Alarm Event 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 Online Help* file
`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 description for the `Config_Alarm_Deadband` function in the *NI-DAQ Function Reference Online Help* file.

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"
```

Set up your program flow like this:

1. Set the properties of the Analog Alarm Event control. Next configure the acquisition or generation operations using the appropriate NI-DAQ functions.
2. Set the `Enabled` property of the Analog Alarm Event control to 1 (True).
3. Invoke the `AnalogAlarmEvent`*`n`*`.Refresh` method to set the Analog Alarm Event in the NI-DAQ driver. Each subsequent invocation of `AnalogAlarmEvent`*`n`*`.Refresh` deletes the old Analog Alarm Event and sets a new one with the current set of properties.
4. Start an asynchronous data acquisition operation.
5. Call any one of the four following procedures:
   - `AnalogAlarm_HighAlarmOn`
   - `AnalogAlarm_HighAlarmOff`
   - `AnalogAlarm_LowAlarmOn`
   - `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 was over 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 copying and pasting a control that already exists on the form. Each individual element in the control array is then 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 is then 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 custom control, 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 control in a Visual Basic program. A working knowledge of Visual Basic is assumed; otherwise, this example is complete, except for error checking:

1. To use the GeneralDAQEvent control, you must first include the proper control into your project.
   - If you are using Visual Basic 4.0 (32-bit), select the **Tools»Custom Controls** option, and select the **National Instruments GeneralDAQEvent** custom control.
   - If you are using Visual Basic 5.0, select the **Project»Components** option, and select the **National Instruments GeneralDAQEvent** custom control. In either version, if you do not find the custom control listed, click on the **Browse** button and find the custom control in the NI-DAQ subdirectory under your Windows directory.

2. To place the GeneralDAQEvent control into your form, go to the tool box window and select the GeneralDAQEvent tool, labelled *DAQ EVENT*.
3. Click somewhere on the form, and while holding down the mouse button, drag the mouse to place the control onto the form. You will see a small icon, which does not appear in run time.
4. To set up a DAQ Event that notifies you after every *n* scans (DAQ Event #1), unless you decide to make *n* very large, you can 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 Online Help* file 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
```

5. Next, start an 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)
```

6. Next, define what to do when the DAQ Event occurs. In this example, we can easily update a text box upon every 1,000 scans and also when the whole acquisition is complete. Place a text box on your form. It is automatically named *Text 1*.

7. 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
```

8. 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
```

9. Run the program. Because you are not going to display the data onto a graph, it does not matter what the data is; however, when you click on the **Click Me!** button, the text box should 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 Online Help* file for guidance.
10. 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 high-level, low-level, and low-level double buffered functions.

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.

One-Shot Analog Input Functions

Single-Channel Analog Input Functions

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.

Refer to the *NI-DAQ Function Reference Online Help* file to determine which functions your device supports.

`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 *`DAQ_Config`* in the *NI-DAQ Function Reference Online Help* file for information on enabling external conversions.
`AI_Clear`	Clears the analog input circuitry and empties the FIFO memory.
`AI_Change_Parameter`	Selects a specific parameter setting for the analog input section or analog input channel. Use this to set the coupling for AI channels.
`AI_Configure`	Informs NI-DAQ of the input mode (single-ended or differential), input range, and input polarity selected for the device. Use this function if you change the jumpers affecting the analog input configuration from their factory settings. For the E Series devices which have no jumpers for analog input configuration, this function programs the device for the settings you want. For the E Series devices you can configure the input mode and polarity on a per channel basis. 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 an MIO and AI device and informs NI-DAQ if any AMUX-64T devices are 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_Read_Scan`	Returns readings for all analog input channels selected by `Scan_Setup`.
`AI_Read_VScan`	Returns readings in volts for analog input channels selected by `Scan_Setup`.
`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 Tips

All of the NI-DAQ functions described in this section are for nonbuffered single-point analog input readings. For buffered data acquisition, consult the *Data Acquisition Functions* section later in this chapter.

Two of the `AI` functions are related to device configuration. If you have changed the device jumper settings from the factory-default settings or want to reprogram the E Series devices, 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. 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. 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* file (nidaqex.hlp) to find a related example.

When using SCXI as a front end for analog input to the 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. Use the AI_Read function to obtain the unscaled data, then call the SCXI_Scale function using both the SCXI module gain and the DAQ device gain.

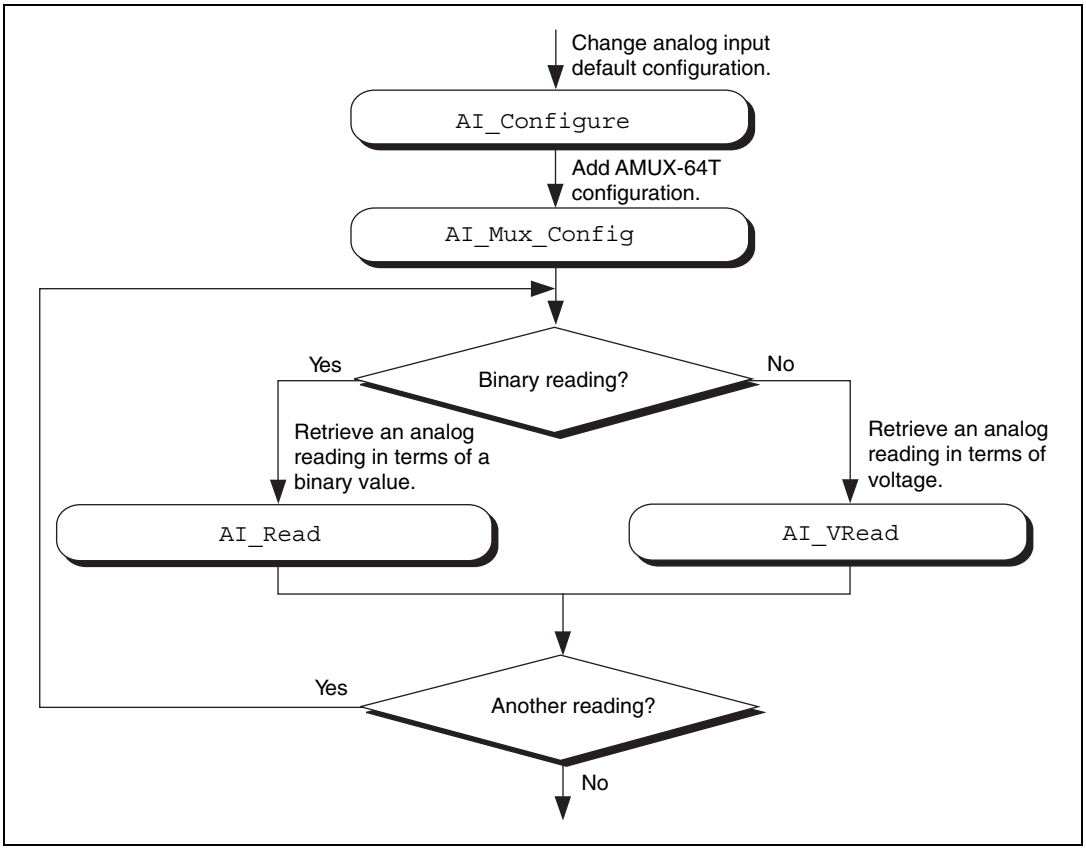


Figure 3-1. Single-Point Analog Reading with Onboard Conversion Timing

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 you are 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.

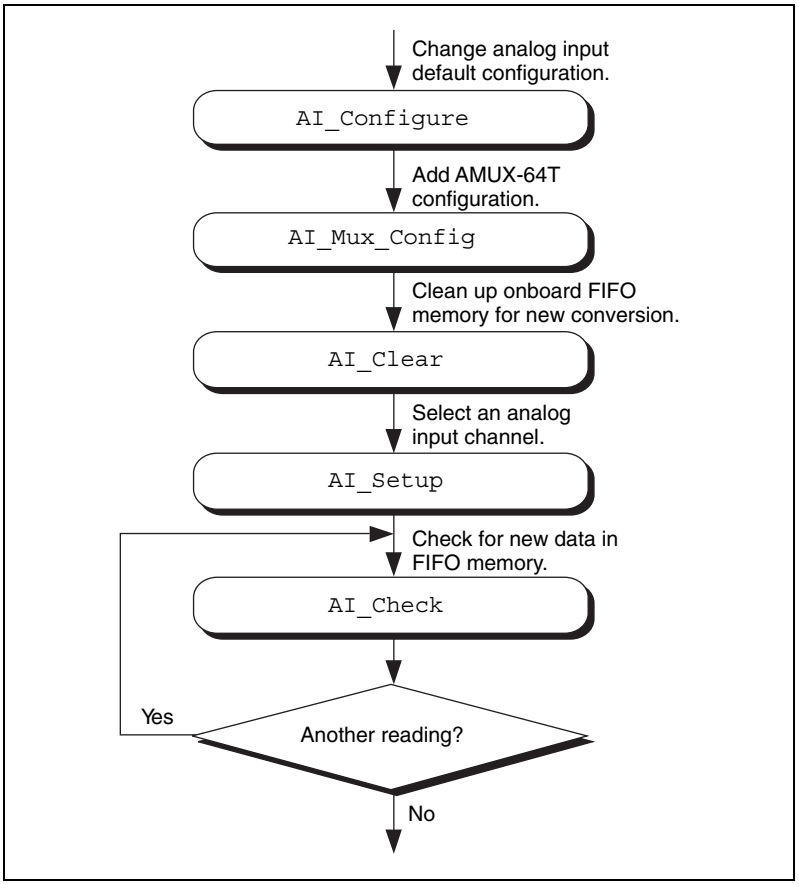


Figure 3-2. Single-Point Analog Reading with External Conversion Timing

Data Acquisition Functions

High-Level Data Acquisition Functions

These high-level data acquisition functions are synchronous calls that acquire data and return when data acquisition is complete.

Refer to the *NI-DAQ Function Reference Online Help* file to determine which functions your device supports.

`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.
`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.

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.
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 it or until an acquisition error has occurred.

Low-Level Data Acquisition Functions

These functions are low-level primitives used for setting up, starting, and monitoring asynchronous data acquisition operations.

Refer to the *NI-DAQ Function Reference Online Help* file to determine which functions your device supports.

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_Set_Clock	Sets the scan rate for a group of channels.
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 PFI 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.
Lab_ISCAN_Start	Initiates a multiple-channel scanned data acquisition operation and stores its input in an array.
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.

`SCAN_Start` Initiates a multiple-channel scanned data acquisition operation, with or without interval scanning, and stores its input in an array.

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.

Refer to the *NI-DAQ Function Reference Online Help* file to determine which functions your device supports.

`DAQ_DB_Config` Enables or disables double-buffered data acquisition operations.

DAQ_DB_HalfReady	Checks if 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 Tips

Lab and 1200 Device 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 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 if 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 only available for data acquisition 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 Device, 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 measurable by the MIO and AI devices with an external multiplexer device (AMUX-64T). Refer to the *AMUX-64T External Multiplexer Devices* chapter 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 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 comprising each building block vary. Typical applications can include the NI-DAQ functions in each of their four building blocks.

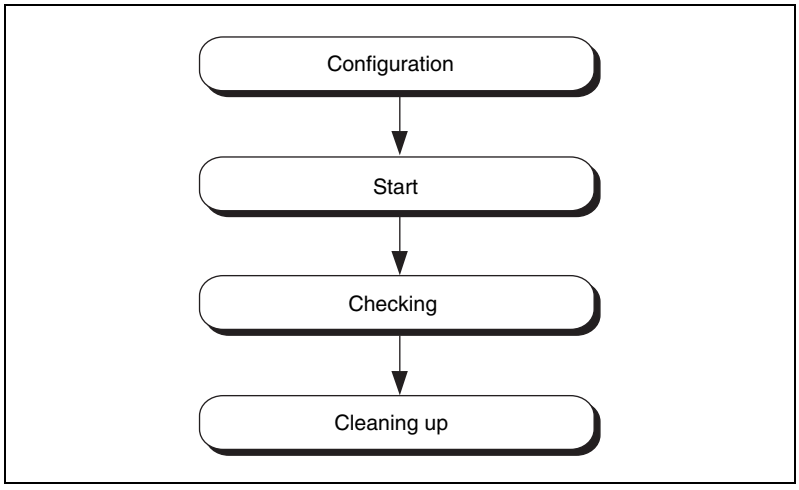


Figure 3-3. Buffered Data Acquisition Basic Building Blocks

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.

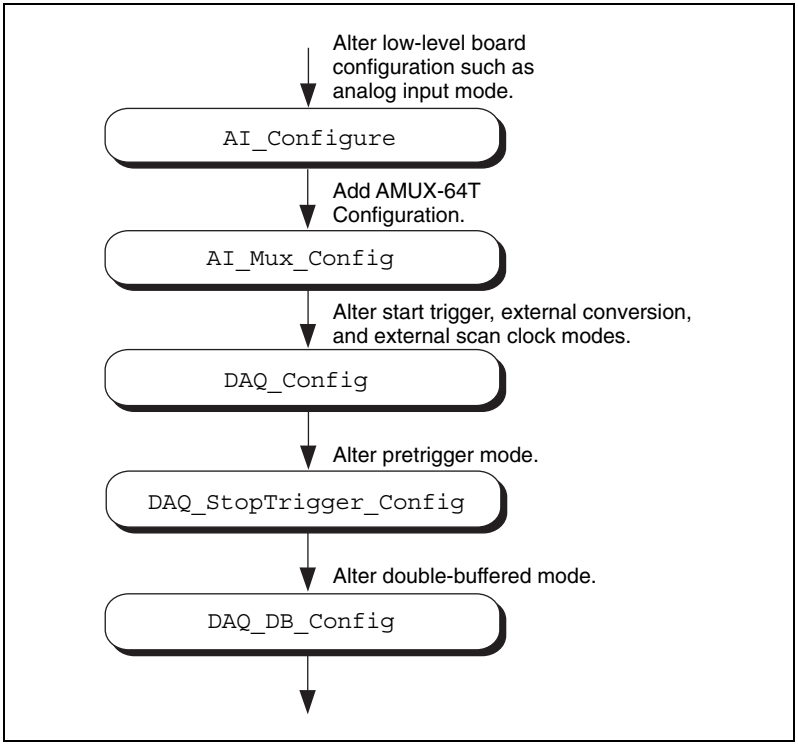


Figure 3-4. Buffered Data Acquisition Application Building Block 1, Configuration

NI-DAQ records the device configurations and the default configurations. (See the `Init_DA_Brds` description in the *NI-DAQ Function Reference Online Help* file 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, AI, and DSA 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, AI, and DSA 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.

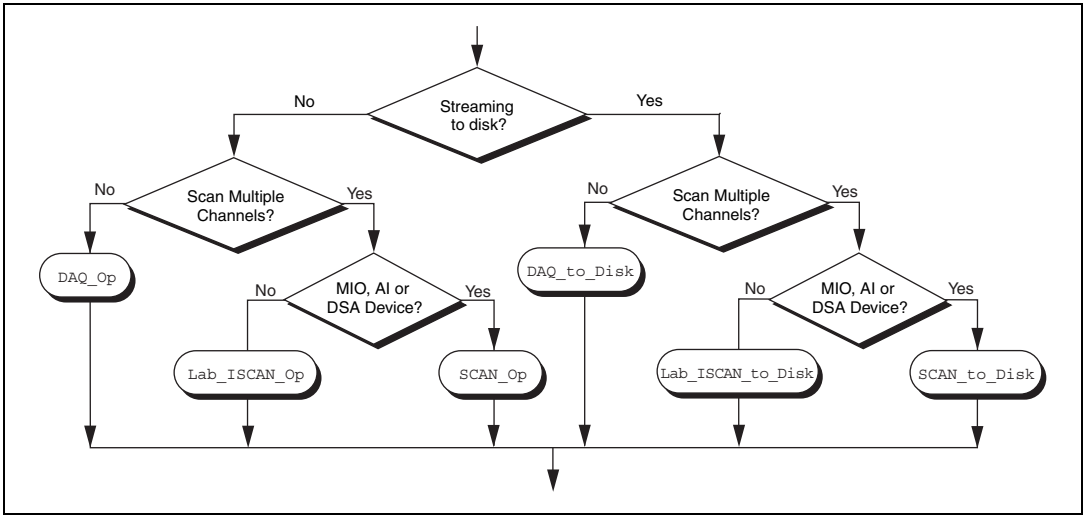


Figure 3-5. Buffered Data Acquisition Application Building Block 2, Start

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 DAQ_Start, SCAN_Start, and Lab_ISCAN_Start. Figures 3-6 and 3-7 show how the start calls make up building block 2 for different devices.

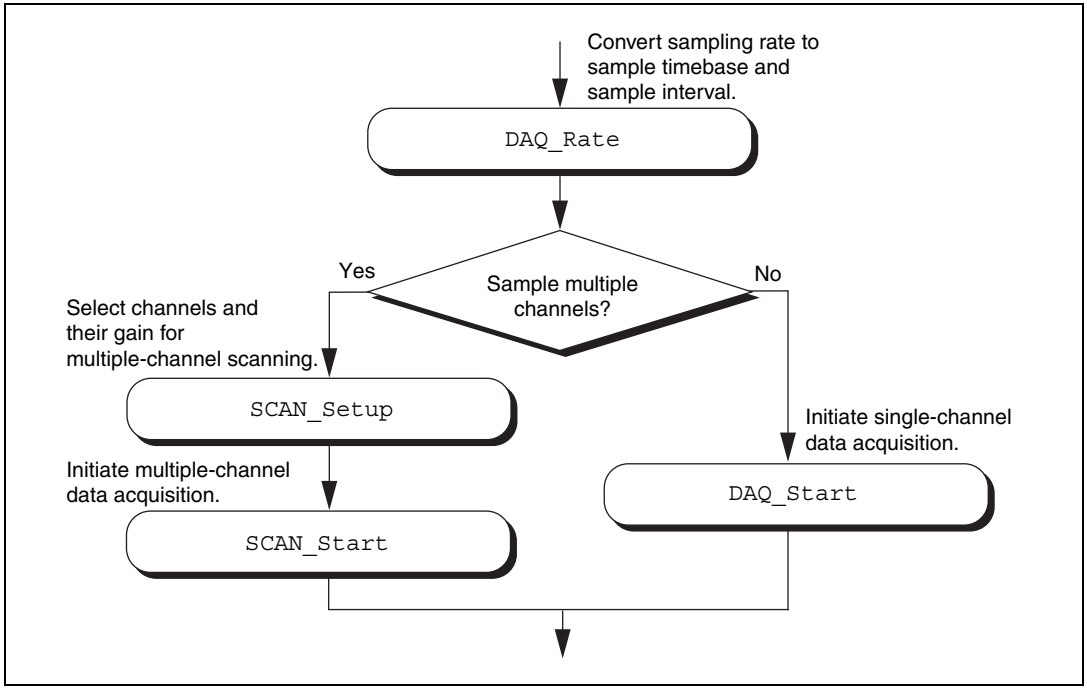


Figure 3-6. Buffered Data Acquisition Application Building Block 2, Start, for the MIO, AI, and DSA Devices

For DSA devices, substitute DAQ_Set_Clock for DAQ_Rate in Figure 3-6. DAQ_Rate will not produce the correct clock settings for DSA devices.

If your device works with multirate scanning, you can use SCAN_Sequence_Setup instead of SCAN_Setup in building block 2.

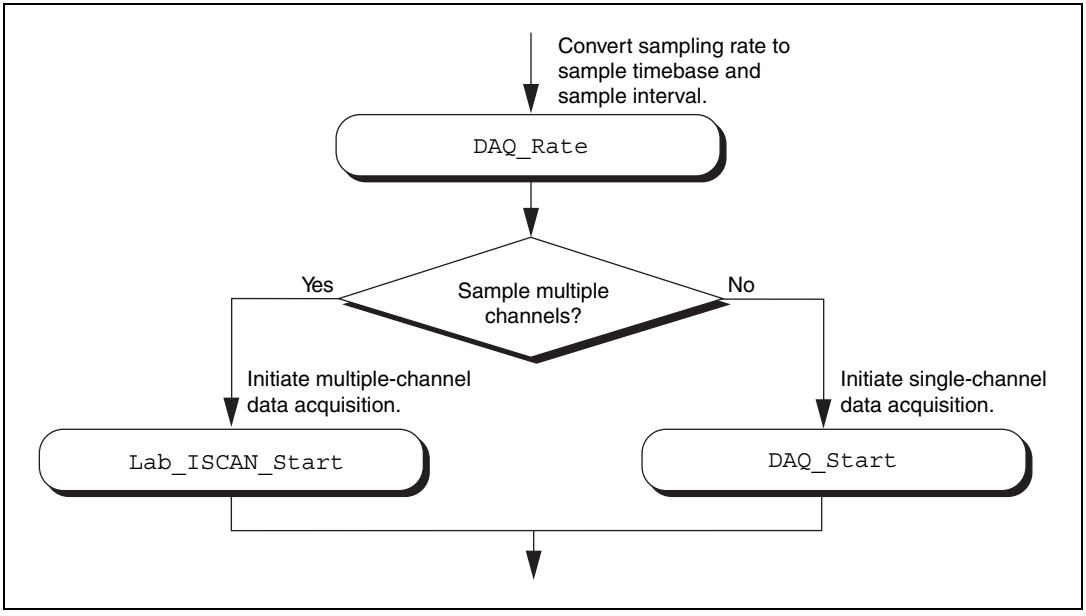


Figure 3-7. Buffered Data Acquisition Application Building Block 2, Start, for the 516 Devices, DAQCard-500/700, Lab and 1200 Devices, and LPM Devices

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

Building Block 3: Checking

DAQ_Check and Lab_ISCAN_Check, shown in Figures 3-8 and 3-9, 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 periodically monitor the data acquisition process.

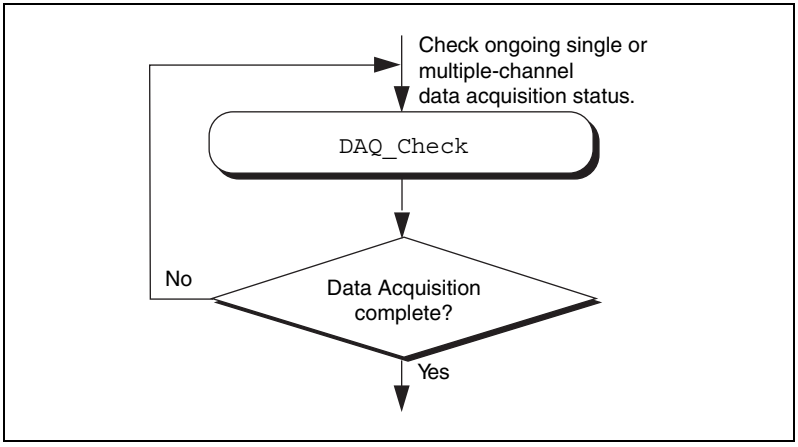


Figure 3-8. Buffered Data Acquisition Application Building Block 3, Checking, for the MIO, AI, and DSA Devices

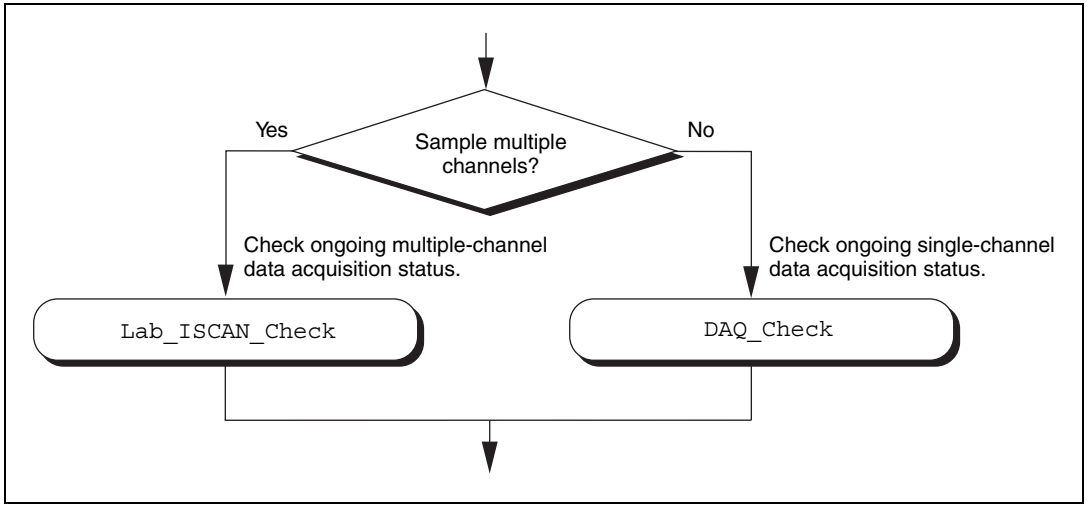


Figure 3-9. Buffered Data Acquisition Application Building Block 3, Checking, for the 516 Devices, DAQCard-500/700, Lab and 1200 Devices, and LPM Devices

However, if the information provided by DAQ_Check is not sufficient, 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 can also retrieve a portion of the acquired data. With the double-buffered functions, you can retrieve half of the data buffer at a time. Double-buffered functions are very useful when your application has a real-time strip chart displaying the incoming data.

Building Block 4: Cleaning up

The purpose of this building block is to stop the data acquisition and free any system resources (such as DMA channels) used for the data acquisition. DAQ_Clear is the only function needed for this building block and is automatically called by the check functions described in the previous building block when the data acquisition is complete. Therefore, you can eliminate this last building block if your application continuously calls the previously described check functions until the data acquisition is complete.

Note DAQ_Clear does not alter the device configurations made by building block 1.

Double-Buffered Data Acquisition

The double-buffered (DAQ_DB) data acquisition functions return data from an ongoing data acquisition without interrupting the acquisition. These functions use a double, or circular, buffering scheme that permits half buffers of data to be retrieved and processed as the data becomes available. By using a circular buffer, you can collect an unlimited amount of data without needing an unlimited amount of memory. Double-buffered data acquisition is useful for applications such as streaming data to disk and real-time data display.

Initiating double-buffered data acquisition requires some simple changes to the first and third basic building blocks, Configuration and Checking, respectively.

In building block 1, turn on double-buffered mode data acquisition through the DAQ_DB_Config call. 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 3-10 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 4, *NI-DAQ Double Buffering*.

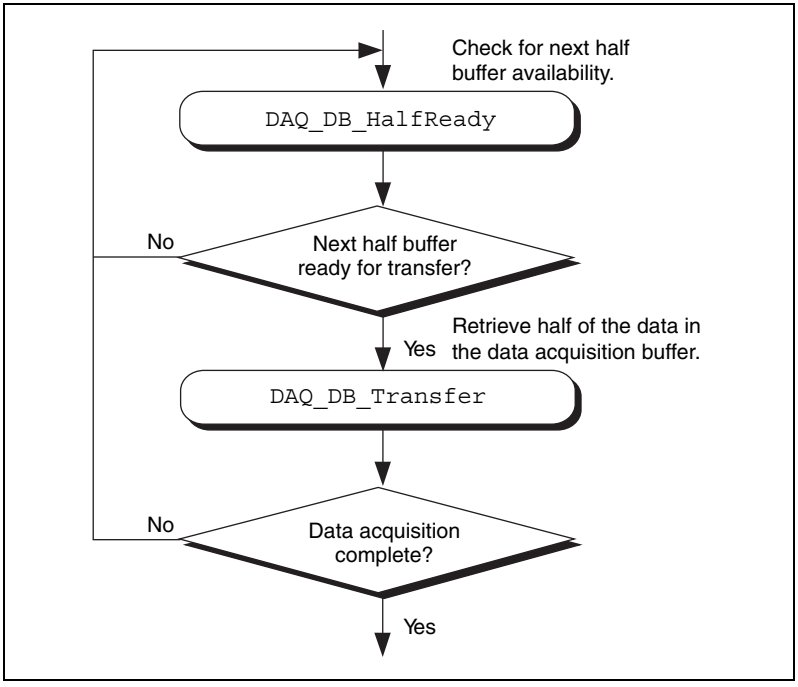


Figure 3-10. Double-Buffered Data Acquisition Application Building Block 3, Checking

Multirate Scanning

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 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, 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 3-11 shows how to use the multirate scanning functions in conjunction with other NI-DAQ functions.

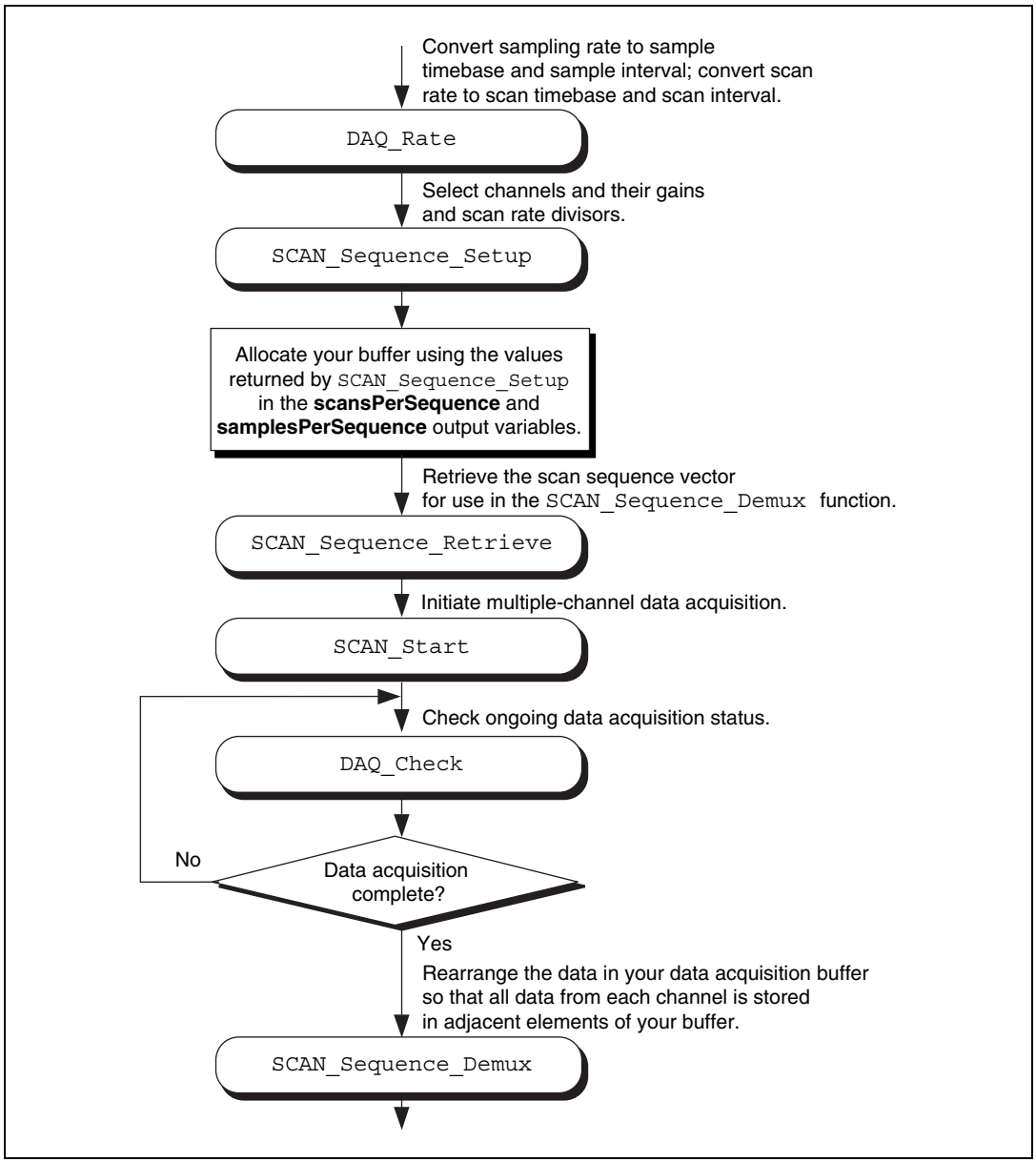


Figure 3-11. Multirate Scanning

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 Use the SCXI functions described later in this chapter for the SCXI-1124 analog output module.

One-Shot Analog Output Functions

Use the Analog Output functions to perform single D/A conversions with analog output devices.

Refer to the *NI-DAQ Function Reference Online Help* file to determine which functions your device supports.

AO_Change_Parameter	Selects a specific parameter setting for the analog output section or analog output channel. These parameters might be data transfer conditions, filter settings, or similar device settings.
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. Use this function if you have changed the jumper settings affecting analog output range and polarity from their factory settings. Also use this function to change the analog output settings on devices without jumpers.
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 Tips

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. AO_VWrite is the equivalent of a call to AO_VScale followed by a call to AO_Write. Figure 3-12 illustrates the equivalency.

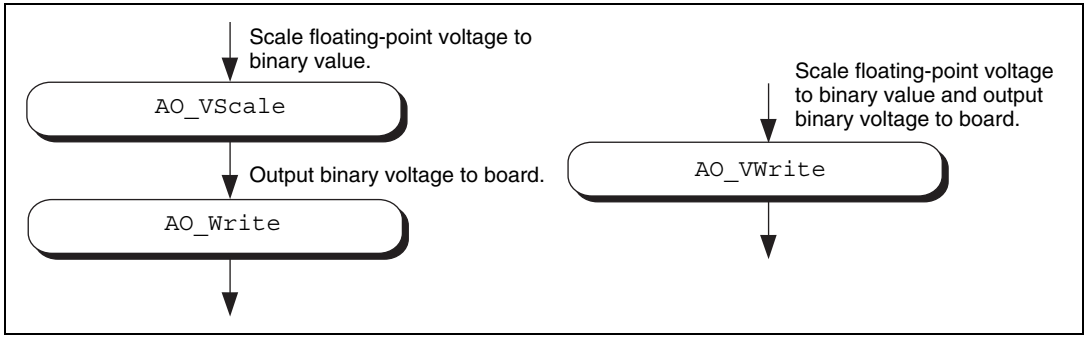


Figure 3-12. Equivalent Analog Output Calls

The following applications are shown using AO_VWrite. However, substituting the equivalent AO_VScale and AO_Write calls will not change the results.

Simple Analog Output Application

Figure 3-13 illustrates the basic series of calls for a simple analog output application.

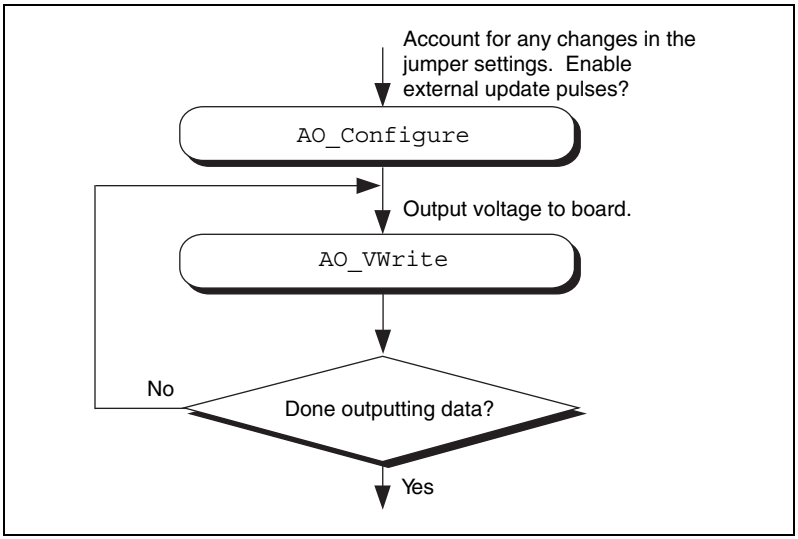


Figure 3-13. Simple Analog Output Application

The call to `AO_Configure` in Figure 3-13 must be made only if you have changed the jumper settings of an MIO, AT-AO-6/10, or Lab-PC+ device. You also might call `AO_Configure` to enable external updating of the voltage. When you select external update mode, voltages written to the device are not output until you apply a pulse 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 E Series device or 671*X* device. You can simultaneously change the voltages at all the analog output channels. The final steps in Figure 3-13 form a simple loop. New voltages are output until the data ends.

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 3-14 illustrates a modified version of the flowchart in Figure 3-13.

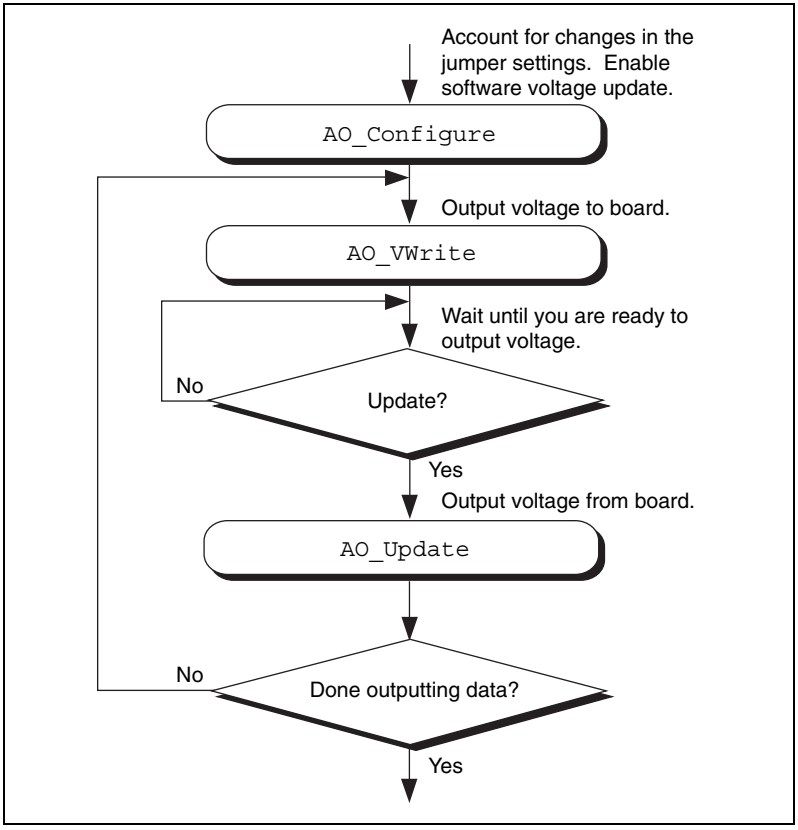


Figure 3-14. Analog Output with Software Updates

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 3-13.

Note Implement buffered analog output using the Waveform Generation (`WFM`) functions.

Waveform Generation Functions

Use the Waveform Generation (WFM) functions to perform buffered analog output operations with the MIO devices, 671*X* devices, AT-AO-6/10 devices, and Lab and 1200 analog output devices.

Refer to the *NI-DAQ Function Reference Online Help* file to determine which functions your device supports.

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.

`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 possible 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, all analog output channels for the Lab and 1200 analog output, 671*X* 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 671*X* devices, E Series devices, and AT-AO-6/10, this function also enables or disables FIFO mode waveform generation.
`WFM_Rate`	Converts a waveform generation update rate into the timebase and update-interval values needed to produce the 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. The function uses the current analog output configuration settings to perform the conversions.
WFM_Set_Clock	Sets an update rate for a group of channels.

Waveform Generation Application Tips

This section outlines a basic explanation of constructing an application with 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 3-15 illustrates the ordinary series of calls for a basic waveform application.

The first step of Figure 3-15 calls WFM_Scale. The WFM_Scale function converts floating-point voltages to integer values, thus producing 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 it. If the number of iterations is nonzero, WFM_Op does not return until the waveform generation is done and all cleanup work is complete. 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. WFM_Rate only supports some devices. Because it does not have a **device number** parameter, it cannot return an error if you use it with a non-supported device. See the *NI-DAQ Function Reference Online Help* file for supported devices.
4. WFM_ClockRate or WFM_Set_Clock to set the update rate (see the *NI-DAQ Function Reference Online Help* file to find out which function supports your device).
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 3-15 shows how 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, CLEAR halts any ongoing waveform generation.

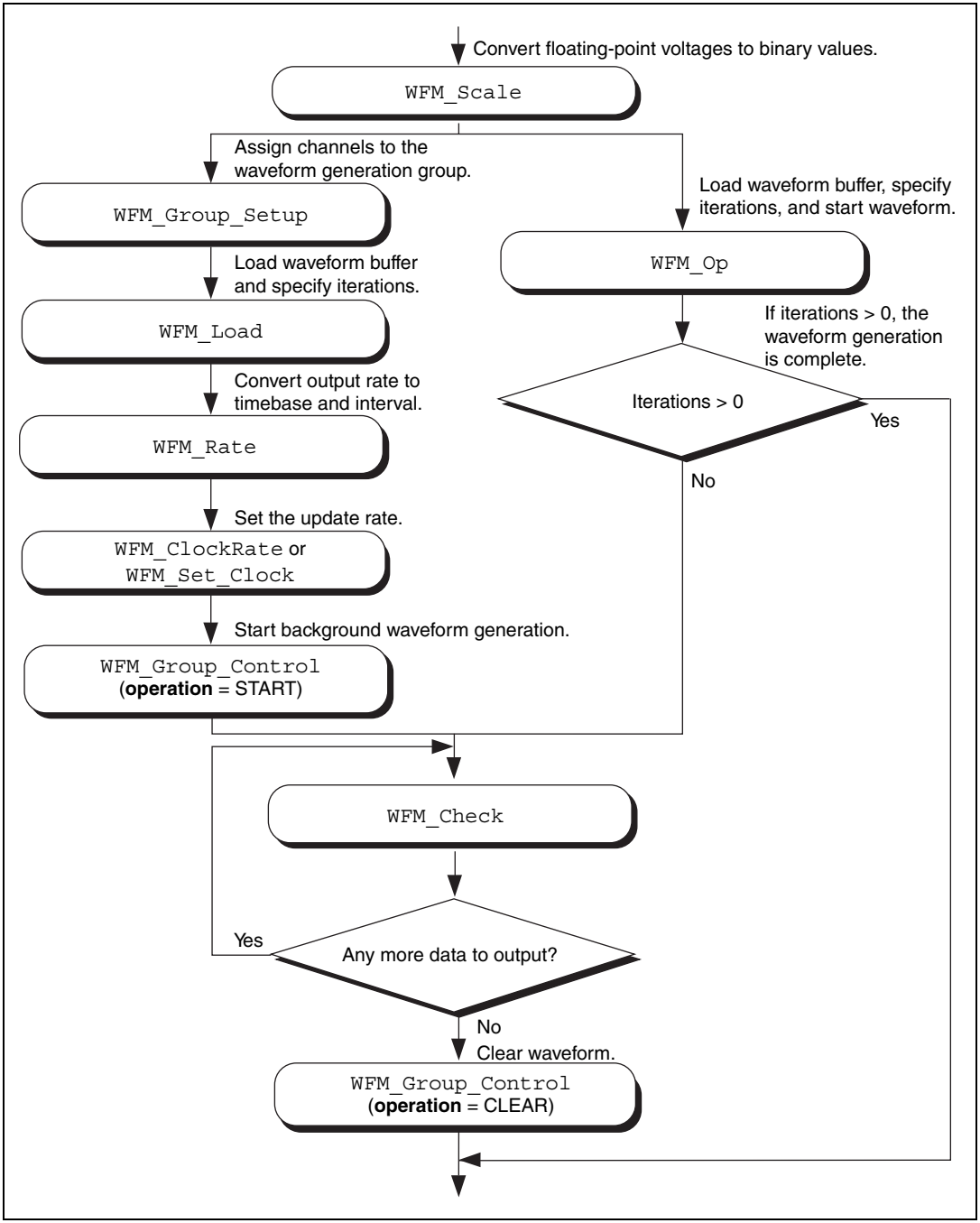


Figure 3-15. Basic Waveform Generation Application

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 3-16 illustrates the ordinary series of calls for a basic waveform application with pauses.

The first step of Figure 3-16 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, 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 the *NI-DAQ Function Reference Online Help* file for further details.

Your application is now ready to start a waveform generation. Call `WFM_Group_Control` (**operation=**`START`) to start the waveform generation in the background. `WFM_Group_Control` will return to your application after the waveform generation begins.

The next step in Figure 3-16 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`). `PAUSE` 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`). `RESUME` 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`). `CLEAR` performs all the necessary cleanup work after a waveform generation. Additionally, `CLEAR` halts any ongoing waveform generation.

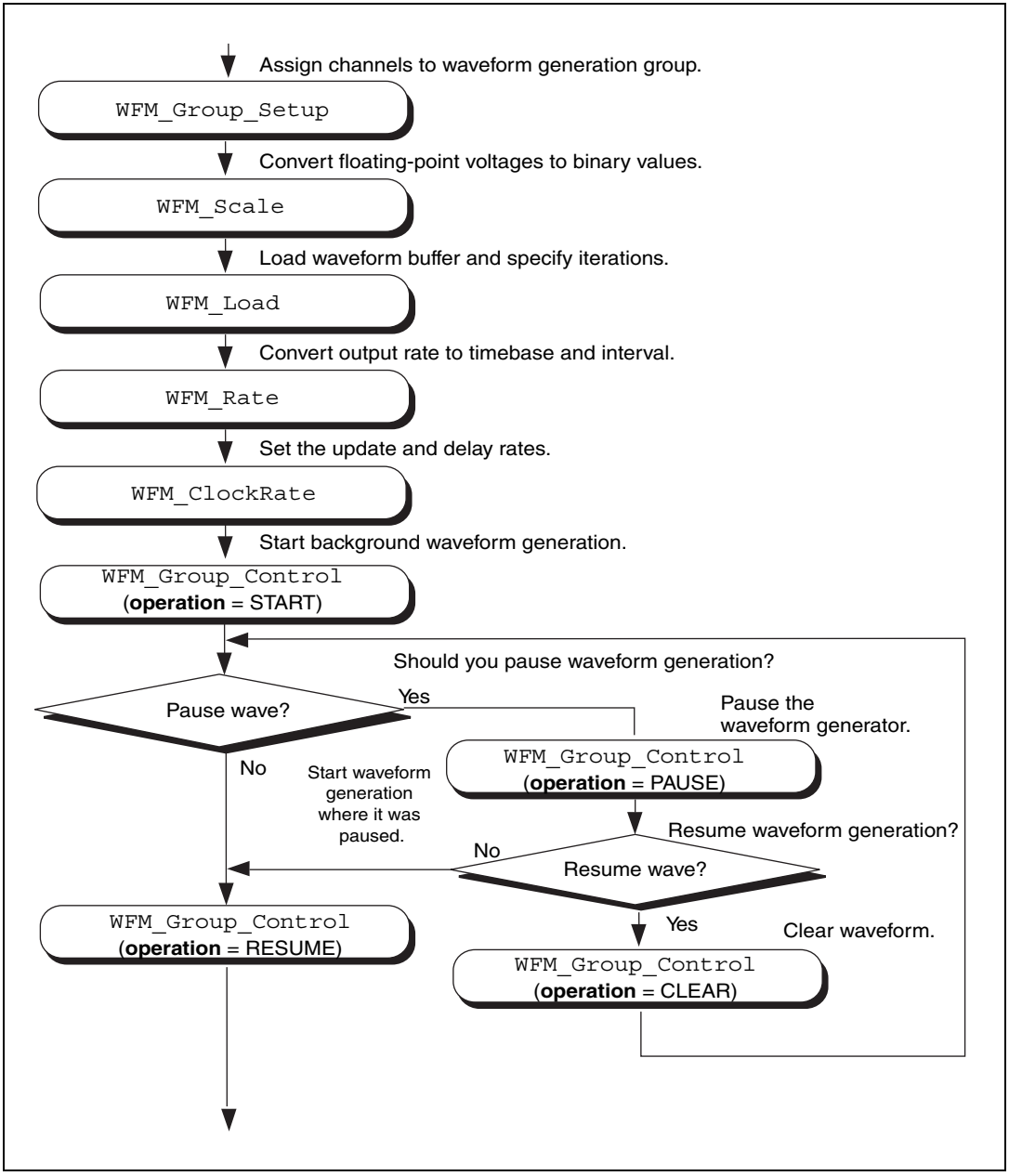


Figure 3-16. Waveform Generation with Pauses

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 4, *NI-DAQ Double Buffering*. Figure 3-17 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 3-17.

Although every step is not in 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 is to call the following sequence of functions—`WFM_Group_Setup` (only required on the AT-AO-6/10), `WFM_Load`, `WFM_ClockRate` or `WFM_Set_Clock`, `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 and WFM_Set_Clock` functions (see the *NI-DAQ Function Reference Online Help* file for the function that supports your device) assign an update rate to a group of analog output channels. Calling `WFM_Group_Control` (**operation**=`START`) starts the background waveform generation. `WFM_Group_Control` returns to your application after the waveform generation begins.

After the operation begins, you can perform unlimited transfers to the circular waveform buffer. To transfer data to the circular buffer, call the `WFM_DB_Transfer` function. After you call the function, 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 transfer progress. 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, CLEAR halts any ongoing waveform generation.

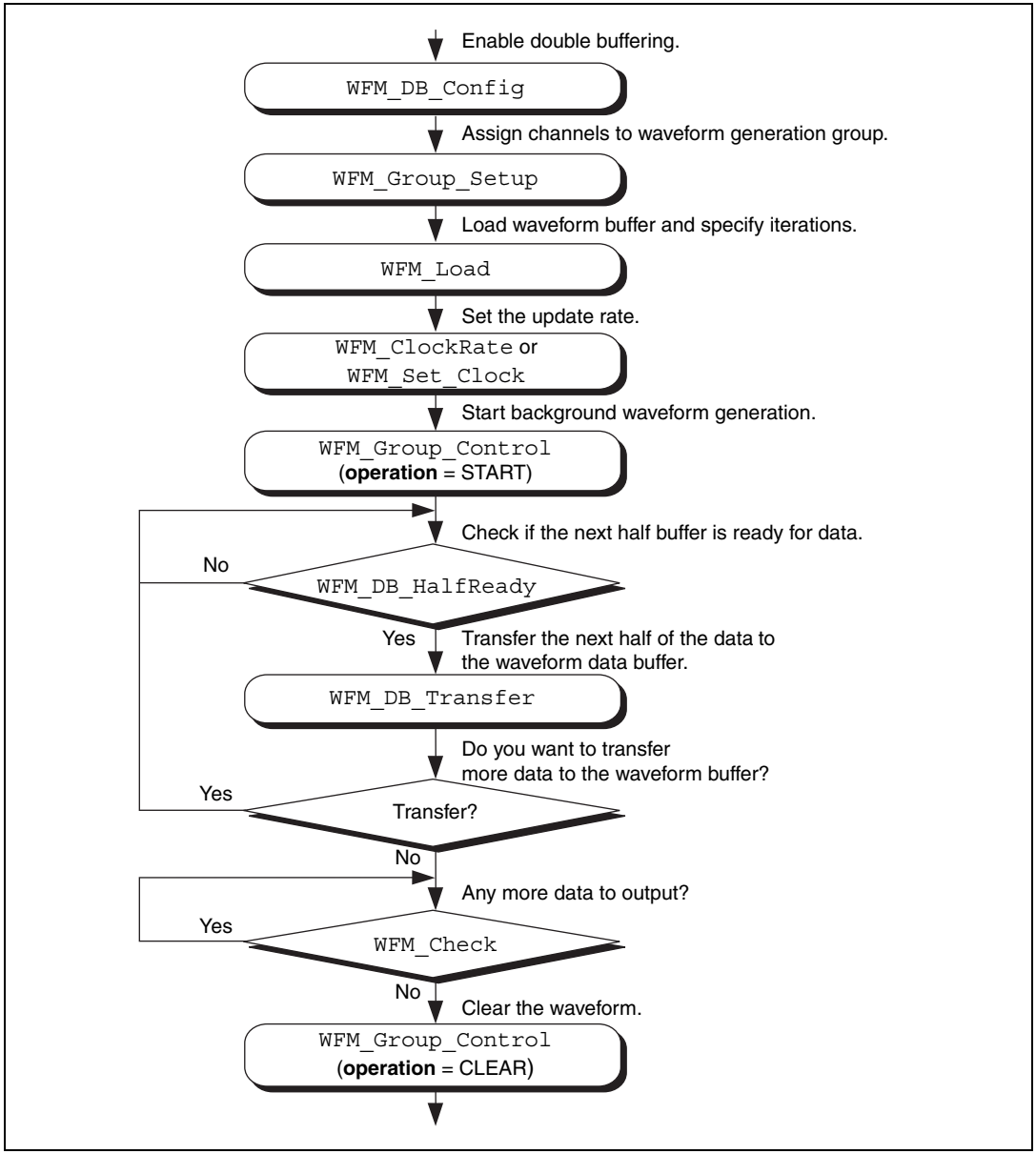


Figure 3-17. Double-Buffered Waveform Generation

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, 6052E, 6053E devices	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-16XE-10, PCI-MIO-16XE-10, PCI-MIO-16XE-50, PCI-6031E (MIO-64XE-10), VXI-MIO-64XE-10, PCI-6110, PCI-6111, 6052E, 6053E devices	—	—	Negative of the reference voltage	Reference voltage
All other MIO devices, 671*X* devices, 622*X* 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	—	—

Note NI 4451 for PCI and NI 4551 for PCI devices use signed, 18-bit binary data left-justified in a 32-bit word. Their output voltage range is ±10 V.

Minimum Update Intervals

The rate at which a device can output analog data is limited by the performance of the host computer. For waveform generation, the limitation is in terms of minimum update intervals. The update interval is the period of time between outputting new voltages. Therefore, the minimum update interval specifies the smallest possible time delay between outputting new data points. In other words, the minimum update interval specifies the fastest rate at which a device can output data. Refer to Chapter 4, *NI-DAQ Double Buffering*, for more information on the minimum update intervals.

Counter Usage

NI 4451 for PCI and NI 4551 for PCI devices use the same counter for both waveform generation and analog input data acquisition. See `WFM_Set_Clock` in the *NI-DAQ Function Reference Online Help* file for an explanation of the restrictions this causes. This counter is separate from the general-purpose counters.

The MIO, 671*X*, 622*X*, and E Series devices use dedicated counters from the DAQ-STC chip for waveform-generation control and timing.

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.

FIFO Lag Effect on the MIO, E Series, AT-AO-6/10, NI 4451 for PCI, NI 4551 for PCI, 622*X*, and 671*X* Devices

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. You see a lag effect for group 1 channels because of the FIFO buffering. 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 is empty.
- 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.
- For 61*XX* devices with onboard memory, data is transferred to the memory in blocks of 32 bytes. Therefore, when you use double-buffered waveform generation that does not end in a 32-byte sample boundary, the last few points will not be output.

With PCI E Series, 622*X* devices, and 671*X* 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 *NI-DAQ Function Reference Online Help* file for details.

Externally Triggering Your Waveform Generation Operation

You can initiate a waveform generation operation from an external trigger signal in much the same manner as for analog input. See the Select_Signal function in the *NI-DAQ Function Reference Online Help* file.

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. Refer to the *NI-DAQ Functions Listed by Hardware Product*

section in the *NI-DAQ Function Reference Online Help* file to find out which digital functions your device supports. The SCXI functions control the SCXI digital and relay modules.

These devices contain a number of digital I/O ports of up to eight digital lines in width. The name *port* 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.

In NI-DAQ, you refer to ports by number. Many digital I/O devices label ports by letter. For these devices, use port number 0 for port A, port number 1 for port B, and so on. For example, the DIO-24 contains three ports of eight digital lines each. Ports 0, 1, and 2 are labeled PA, PB, and PC on the DIO-24 I/O connector. The eight digital lines making up port 0, lines 0 through 7, are labeled PA0 through PA7.

In some cases, you can combine digital I/O ports into a larger entity called a *group*. On the DIO-32F and 653*X* devices, for example, you can assign any of the ports DIOA through DIOD to one of two groups. On the PCI-6115 and PCI-6120, you can also create groups smaller than the port size. For example, both the PCI-6115 and the PCI-6120 have one digital port of eight lines, but you can configure five lines as an input group and three lines as an output group. 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 byte mapping to the digital I/O lines is as follows.

Table 3-6. Byte Mapping 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

Table 3-6. Byte Mapping to Digital I/O Lines (Continued)

Bit Number	Digital I/O Line Number
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, 653*X*, DSA, 671*X*, and 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; 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, 6025E devices, and AT-MIO-16DE-10 (ports 2 and 3 only), DIO-96, Lab and 1200 devices, 653*X*, DIO-32F, PCI-6115 and PCI-6120 devices for handshaking and no-handshaking modes. These two modes have the following characteristics:

- No-handshaking mode—This mode 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—This mode is 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. The handshaking mode for the PCI 6115 and the PCI-6120 is slightly different and is more appropriately called a clocking mode. In the clocking mode, no two-way handshaking signals are generated. Instead, data is latched in or latched out when a pulse from a clock signal is detected.

Note On the 653*X*, DIO-32F, PCI-6115, and PCI-6120 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, 6025E, AT-MIO-16DE-10, DIO-96, and Lab and 1200 Device Groups

You can group together any combination of ports 0, 1, 3, 4, 6, 7, 9, and 10 on the DIO-96, ports 0 and 1 on the DIO-24 and Lab and 1200 devices, and ports 2 and 3 on the 6025E devices 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 Tips* later in this chapter and the `DIG_SCAN_Setup` function description in the *NI-DAQ Function Reference Online Help* file for more details.

DIO-32F and 653*X* Device Groups

On the DIO-32F and 653*X* 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 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 you assign ports 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 the *NI-DAQ Function Reference Online Help* file.

PCI-6115 and PCI-6120 Device Groups

On the PCI-6115 and PCI-6120 devices, port 0 can be broken into two smaller groups. These groups are referred to as group 1 and group 2. One group is configured as an input group and the other is configured as an output group.

Because these devices only have one digital port, a group can consist of any combination of digital lines 0-7. For example, group 1 can be configured as an input group containing lines 0-4 and group 2 as an output group containing lines 5-7. Alternatively, group 1 can be configured as an output group containing lines 1,3,5,7 and group 2 as an input group containing lines 0,2,4,6. A portion of the eight available lines can also be configured for group operations, with the remaining unused lines configured for

immediate digital operations. However, both groups cannot be configured for input or output at the same time.

Although a group can consist of non-contiguous lines, there is no logical grouping of the lines, and an 8-bit value is still passed from the input function or returned from the output function. For example, if a group is configured for output with lines 0,1,2,4, and the group is to be written with all ones, the hex value of 0x17 is passed to the function instead of 0xF.

Group operations are buffered operations only, which means immediate digital operations are not possible with these devices using handshaking. Because all group operations are buffered, there is no support for a status operation indicating when an input operation has been latched to or an output operation latched from the digital lines. Instead, group operations are done from a buffer. In the case of digital output operations, values are written to a buffer and latched onto the digital lines when the appropriate edge from a clock signal is detected. For digital input operations, values are latched from the digital lines to the buffer when the appropriate edge of the clock signal is detected. Either a rising edge or a falling edge of the clock signal can be specified for latching the values. The clock source can be a timing signal used internally or generated external to the board, and it must be routed to one of the RTSI lines.

Digital I/O Functions

The digital I/O (`DIG`) functions perform nonhandshaked digital line and port I/O.

Refer to the *NI-DAQ Function Reference Online Help* file to determine which functions your device supports.

`DIG_In_Line`	Returns the digital logic state of the specified digital input line in the specified port.
`DIG_In_Prt`	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).
`DIG_Out_Line`	Sets or clears the specified digital output line in the specified digital port.

DIG_Out_Prt	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.

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.

Refer to the *NI-DAQ Function Reference Online Help* file to determine which functions your device supports.

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 data transfer from the specified group to memory.
DIG_Block_Out	Initiates an asynchronous data transfer from memory to the specified group.
DIG_Block_PG_Config	Enables or disables the pattern generation mode of buffered digital I/O.
DIG_Grp_Config	Configures the specified group for port assignment, direction (input or output), and size.
DIG_Grp_Mode	Configures the specified group for handshake signal modes.
DIG_Grp_Status	Returns a status word indicating the handshake status of the specified group.
DIG_In_Grp	Reads digital input data from the specified digital group.

DIG_Out_Grp	Writes digital output data to the specified digital group.
DIG_SCAN_Setup	Configures the specified group for port assignment, direction (input or output), and size.
DIG_Trigger_Config	Enables or disables the trigger mode of buffered digital I/O to indicate when to start and stop the data acquisition.

Double-Buffered Digital I/O Functions

The double-buffered digital I/O (DIG_DB) functions perform double-buffered operations during Group Digital I/O operations.

Refer to the *NI-DAQ Function Reference Online Help* file to determine which functions your device supports.

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 possible 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 Change Notification Functions

The Digital Change Notification functions provide messaging for lines and ports on the 652*X* devices. For other boards such as the DIO-24, the `Config_DAQ_Event_Message` function handles event messaging.

`DIG_Change_Message_Config`
Configures 652*X* devices to detect rising or falling edges on input lines and to notify you by generating a message.

`DIG_Change_Message_Control`
Controls the change notification operation of the digital input lines on 652*X* devices.

Digital Filtering Function

The Digital Filtering function provides signal conditioning to filter the inputs of 652*X* devices.

`DIG_Filter_Config` Configures filtering for the input lines on 652*X* devices.

Digital Change Notification Applications with 652*X* Devices

Digital change notification applications automatically detect changes on input lines and notify you or your software by message. These applications may use digital filtering to eliminate signals that may trigger unwanted change notification. Digital filtering can be used alone to condition and debounce input data.

Figure 3-18 illustrates the series of calls needed for change notification on the input data. Figure 3-19 illustrates the series of calls needed for filtering the input data without change notification. Only 652*X* devices can execute change notification and filtering applications using these functions.

Digital Change Detection Applications with 653*X* Devices

For change detection on the 653*X*, see the `DIG_Block_PG_Config` function in the function reference. For message generation on the 653*X* and many other devices, see the `Config_DAQ_Event_Message` function in the function reference.

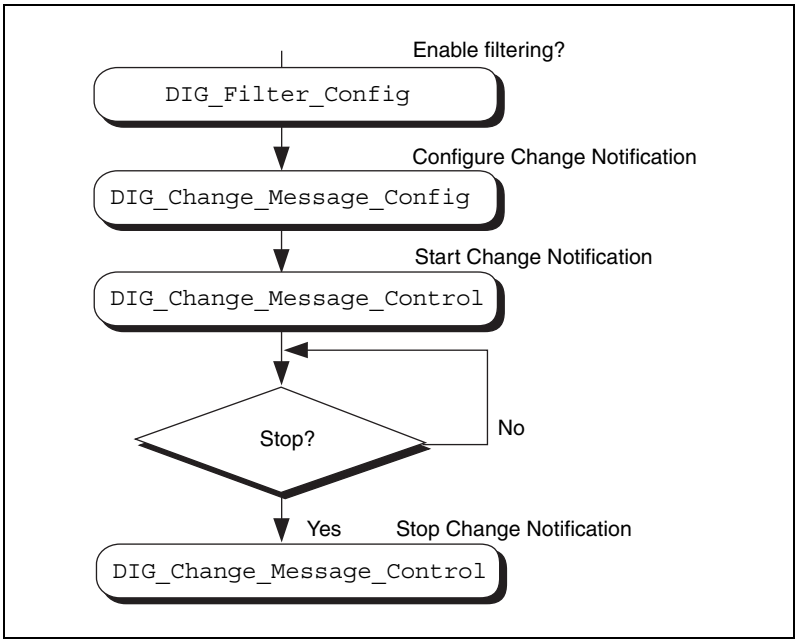


Figure 3-18. Basic Digital Change Notification

To configure change notification, call `DIG_Change_Message_Config`. With `DIG_Change_Message_Config`, you can configure individual digital lines for rising, falling, or rising and falling edge detection. Call `DIG_Filter_Config` to enable filtering on some or all of the lines.

The next step is to start change detection messaging by calling the `DIG_Change_Message_Control` function with the start control code.

To stop change notification, call `DIG_Change_Message_Control` with the stop control code. These steps form the basis of a basic digital change notification application.

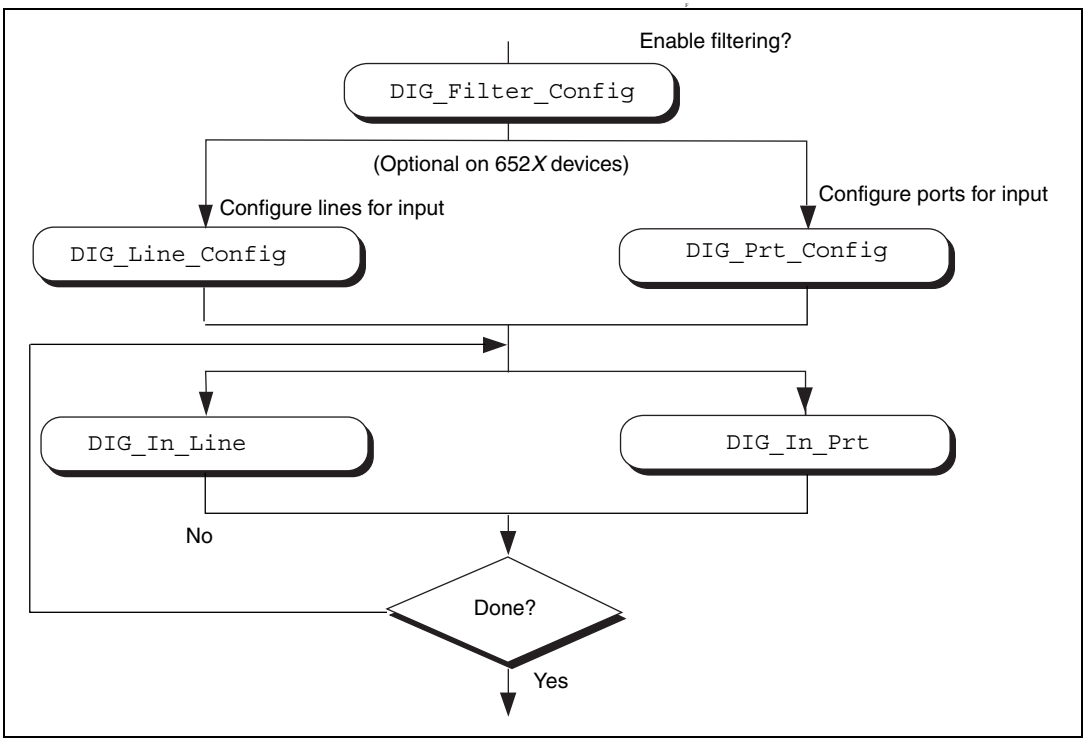


Figure 3-19. Basic Digital Filtering of Input Data Application

The first step is to call `DIG_Filter_Config`, which enables filtering on the specified lines. Next, you can configure either the ports or the lines. For the 652*X* devices, configuring the ports by calling `DIG_Prt_Config` is optional, because the ports have fixed directions. `DIG_Line_Config` is also optional because the lines within the ports have fixed directions.

The next step is to call `DIG_In_Port` to read data from an input port. Call `DIG_In_Line` to read a bit from a line. The final step is to loop back if more data is to be read.

Digital I/O Application Tips

This section gives a basic explanation of how to construct an application using the digital input and output functions. The flowcharts are a quick reference for constructing potential applications from the NI-DAQ function calls.

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 is 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.

You can configure individual port transfers for handshaking or no-handshaking. All AT and PC devices with digital I/O ports can use no-handshaking digital port I/O. DIO-24, 6025E devices, AT-MIO-16DE-10, DIO-96, and Lab and 1200 devices can also execute handshaking digital I/O for using the port I/O functions.

Figure 3-20 illustrates the series of calls for digital port I/O applications with handshaking. Figure 3-21 illustrates the series of calls for digital port I/O applications without handshaking.

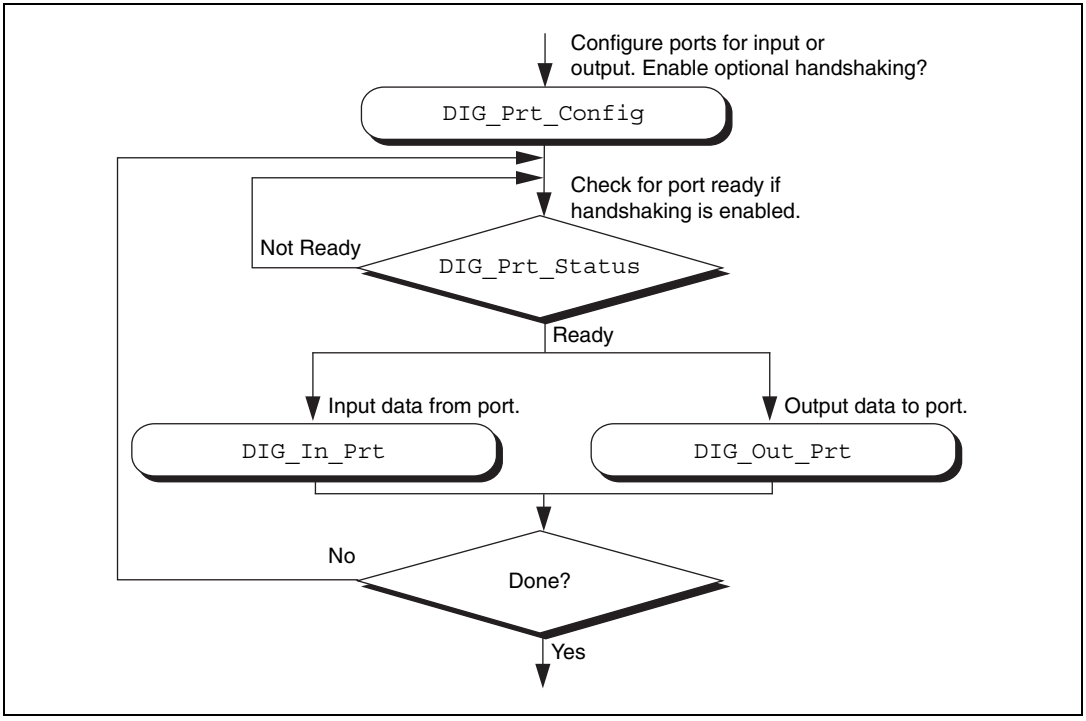


Figure 3-20. Basic Port Input or Output Application with Handshaking

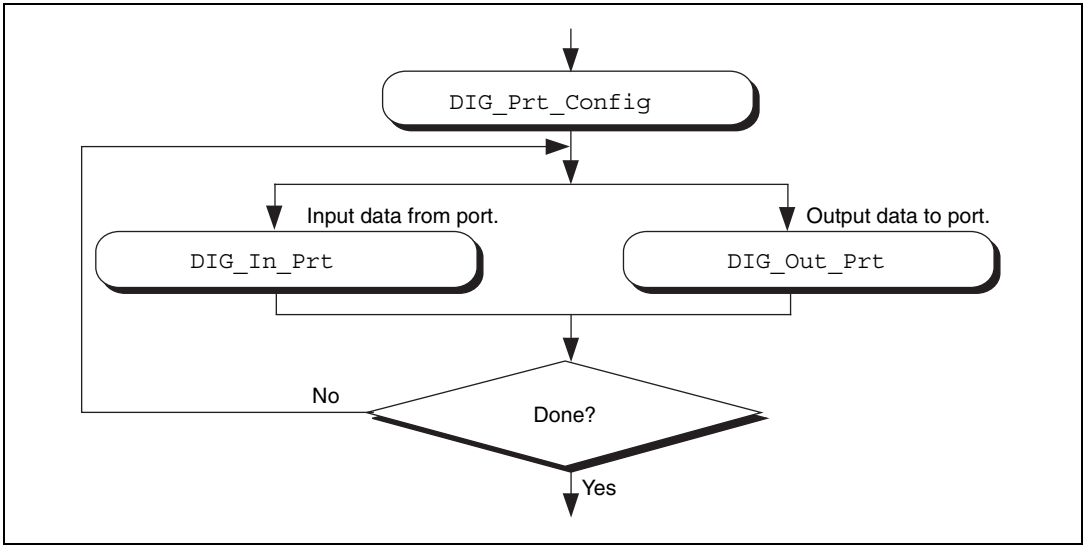


Figure 3-21. Basic Port Input or Output Application without Handshaking

The first step is to call DIG_Prt_Config, with which you configure the individual digital ports for input or output and enable handshaking.

If handshaking is disabled, do not check the port status (step 2 of Figure 3-21). 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_Prt to read data from an input port. Call DIG_Out_Prt 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 can only transfer data in no-handshaking mode.

Figure 3-22 is a flowchart outlining the basic line I/O application.

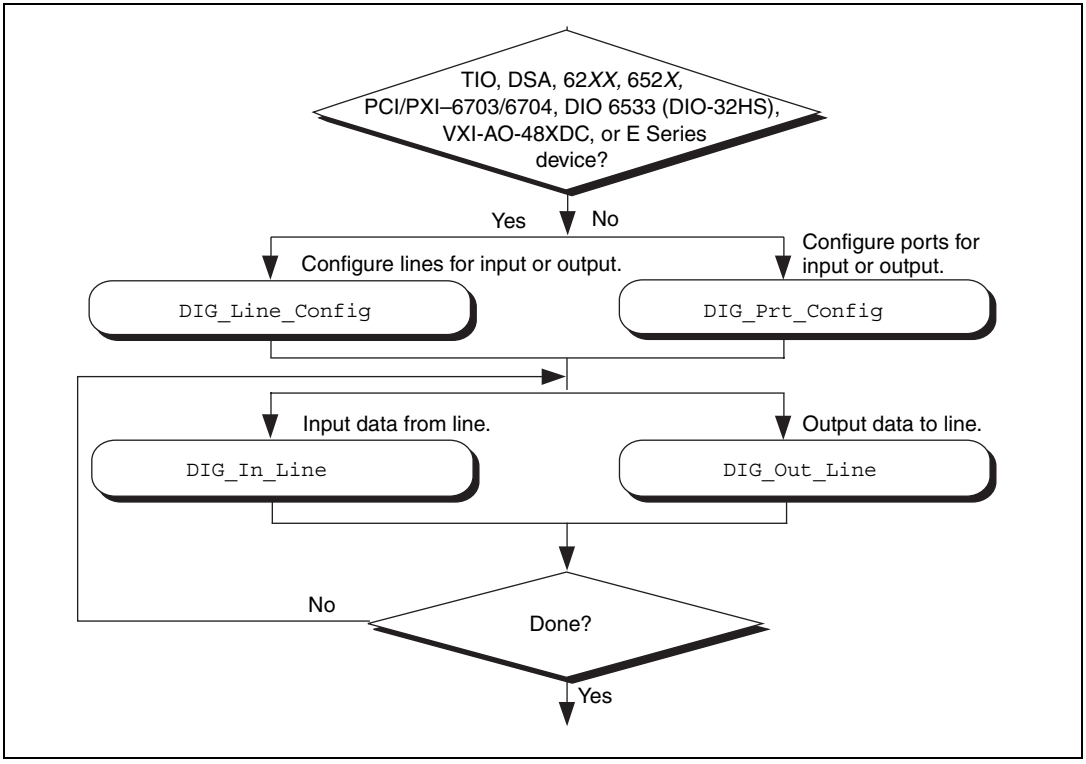


Figure 3-22. Basic Line Input or Output Application

First, configure the digital lines for input or output. You can program 653*X* devices, PC-TIO-10, PCI-6703, PCI-6704, PXI-6703, PXI-6704, 622*X*, 671*X,* 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. 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. For the 652*X* devices it is not necessary to configure a port or line since the line direction is preconfigured.

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, except for the PCI-6115 and the PCI-6120, which can create groups smaller than the port size for digital operations.

Figure 3-23 is a flowchart for group digital applications that input and output data one point at a time. Only the DIO-32F and 653*X* devices can execute group input or output one point at a time.

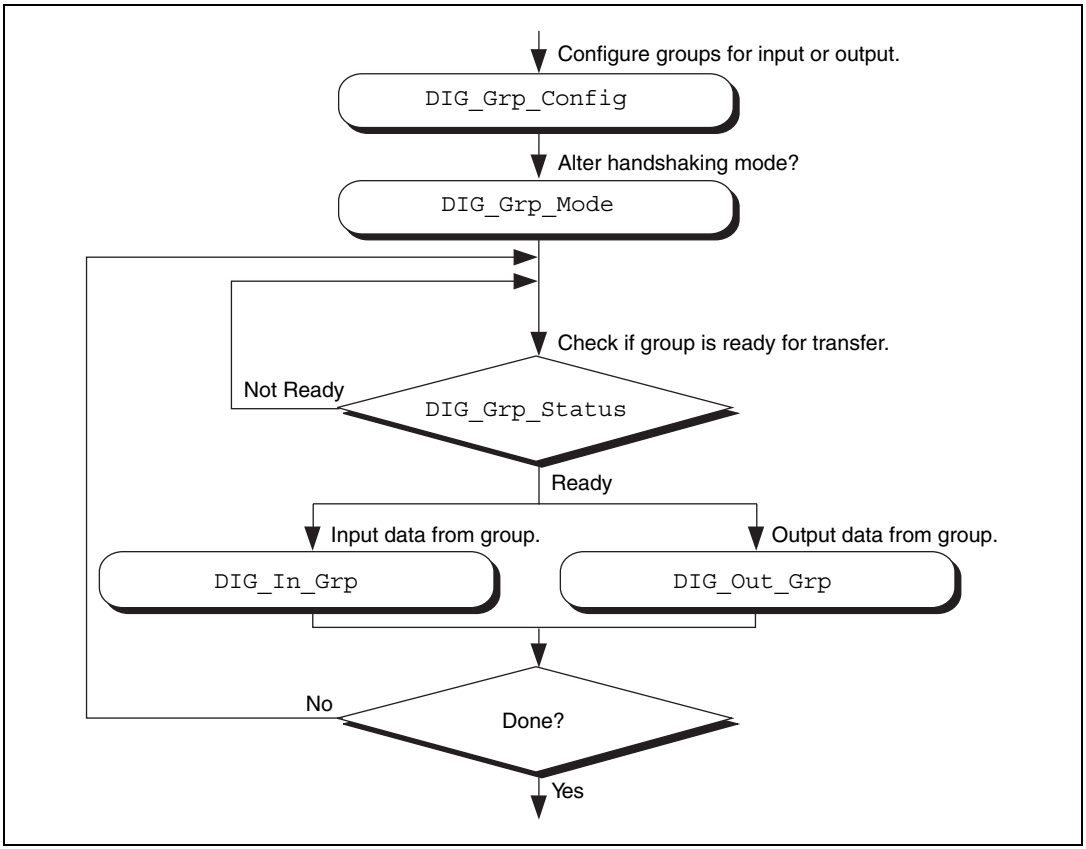


Figure 3-23. Simple Digital Group Input or Output Application

At the start of your application, call `DIG_Grp_Config` to configure the individual digital ports as a group. After the ports are grouped, call `DIG_Grp_Mode` (step 2 of Figure 3-23) to alter the handshaking mode of the DIO-32F and 653*X* devices. The various handshaking modes and the default settings are explained in the `DIG_Grp_Mode` function description.

The next step in your application is to check if the port is ready for a transfer (step 3 of Figure 3-23). To do this, call `DIG_Grp_Status`. If the group status indicates it is ready, call `DIG_Out_Grp` or `DIG_In_Grp` to transfer the data to or from the group.

The final step of the flowchart is to loop back until all of the data has been input or output.

Digital Group Block I/O Applications

NI-DAQ also contains group digital I/O functions, which operate on blocks of data. Figure 3-24 outlines the basic steps for applications that use block I/O.

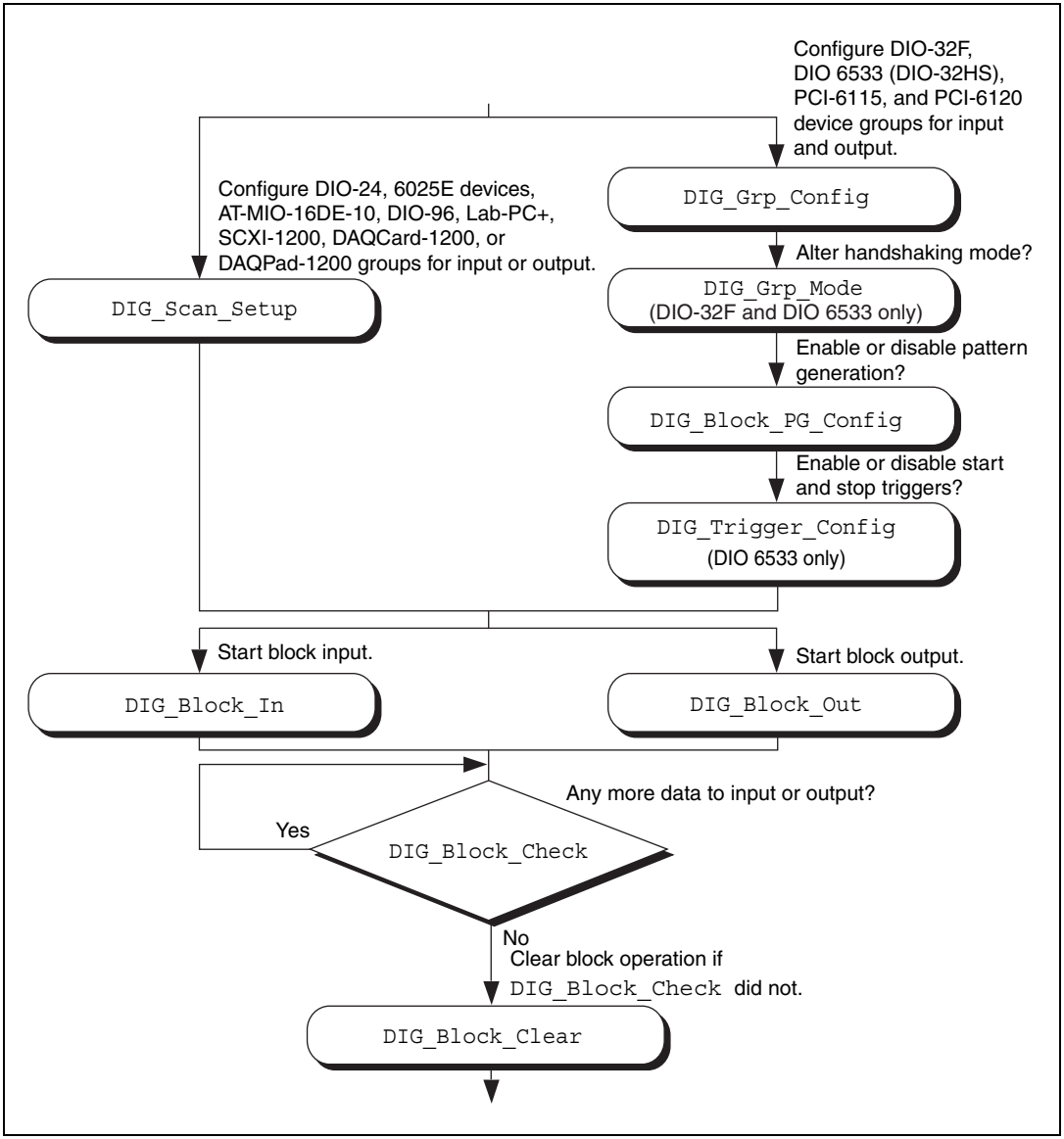


Figure 3-24. Digital Block Input or Output Application

Note The DIO-32F, 653*X*, DIO-24, 6025E devices, AT-MIO-16DE-10, DIO-96, and Lab and 1200 devices all can perform group block operations. However, the DIO-24, 6025E devices, 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 653*X* 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, 653*X,* PCI-6115, or a PCI-6120 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 653*X* device, you can alter the handshaking mode of the group by calling `DIG_Grp_Mode`. For the DIO-32F, 653*X,* PCI-6115, and PCI-6120, you can perform digital pattern generation by calling `DIG_Block_PG_Config`, as shown in Figure 3-24. Pattern generation is simply reading in or writing out digital data at a fixed rate. This is the digital equivalent of analog waveform generation. To enable pattern generation, call `DIG_Block_PG_Config` as shown in Figure 3-24. You cannot handshake with pattern generation, so do not connect any handshaking lines. Refer to the explanation of pattern generation later in this chapter for more information.

The next step for your application, as illustrated in Figure , 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 3-25 outlines the basic steps for digital double-buffered group block I/O applications.

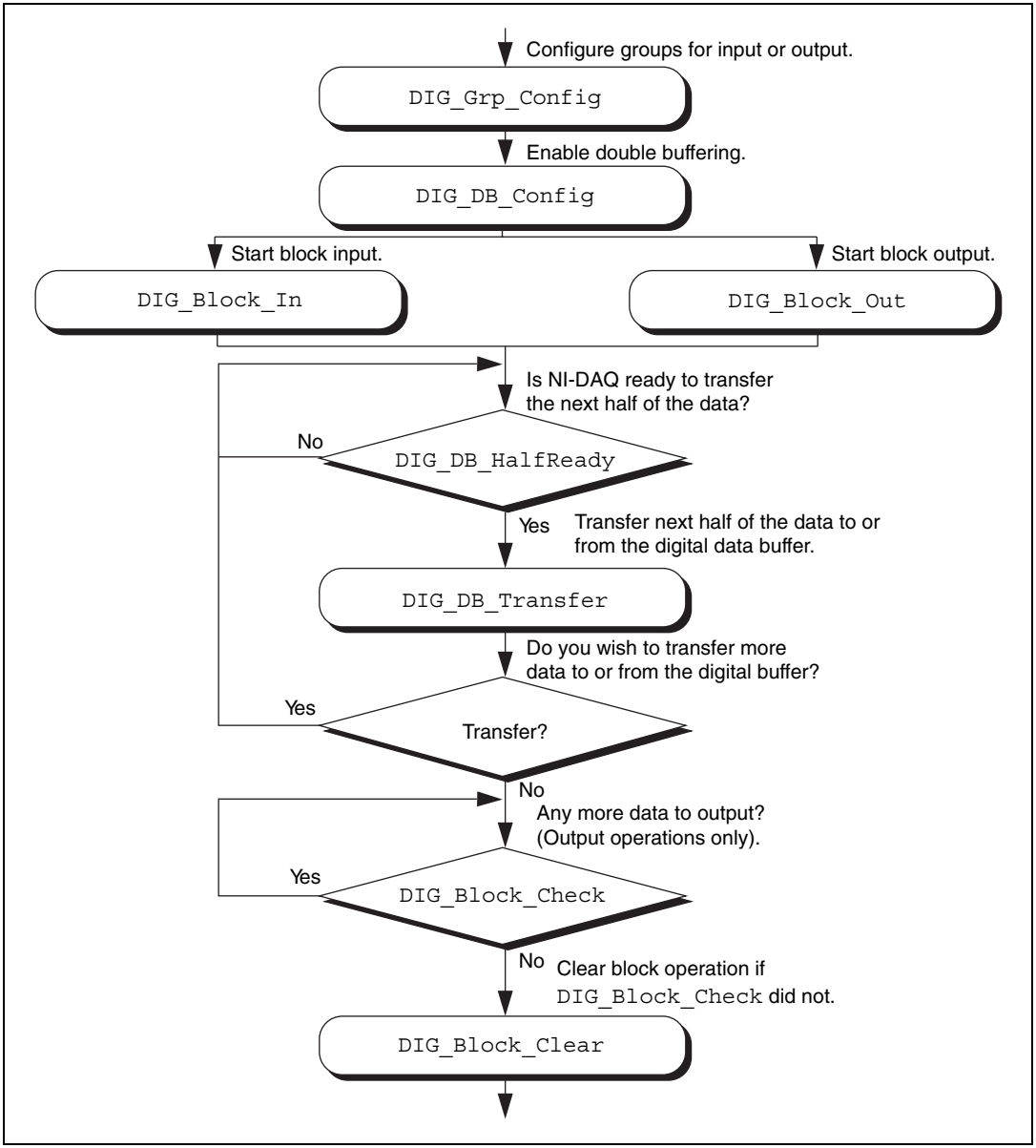


Figure 3-25. Double-Buffered Block Operation

The first step for an application is to call DIG_Grp_Config to configure individual ports as a group. Although the steps have been left out of the diagram, you can alter the handshaking mode and enable pattern generation as shown in Figure 3-24, and explained in the *Digital Group Block I/O Applications* section earlier in this chapter. Next, enable double buffering by calling DIG_DB_Config (second step of Figure 3-25). To start the digital block input or output, call DIG_Block_In or DIG_Block_Out.

After the operation has started, you can perform unlimited 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 can do other processing and check the status later.

After the final transfer, you can call DIG_Block_Check to get the current progress of the transfer. For example, if you are using double-buffered output, NI-DAQ requires some time after the final transfer to actually output the data. In addition, if NI-DAQ completes the block operation prior to a DIG_Block_Check call, DIG_Block_Check automatically calls DIG_Block_Clear to perform cleanup work.

The final step of a double-buffered block operation is to call DIG_Block_Clear, which 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 is complete or if you want to stop the current operation.

Pattern Generation I/O with the DIO-32F, 653*X*, PCI-6115, and PCI-6120 Devices

Use pattern generation for clocked digital I/O when you have a group 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 for the DIO-32F and 653*X* devices has its own onboard counter so that each can simultaneously run in this mode at different rates. 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, 653*X,* PCI-6115, and PCI-6120 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 reprogram increases the minimum request interval (thus decreasing the maximum rate unless you use dual DMA). Refer to Chapter 4, *NI-DAQ Double Buffering*, for more information.

Note For the AT-DIO-32F, `DIG_Block_In` and `DIG_Block_Out` return a warning if it is necessary to reprogram the DMA controller. Also, page boundaries in a buffer that is to be used for 32-bit digital pattern generation cause 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 the *NI-DAQ Function Reference Online Help* file.
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 completes.
8. Because DIG_Block_Clear unaligns the buffer, you can access the digital input pattern generation as you can with an unaligned buffer. 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 4, *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. If the group is configured 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 of half buffers are enabled.

Counter/Timer Function Group

The Counter/Timer function group contains three sets of functions—the General Purpose Counter/Timer (`GPCTR`) functions, the Interval Counter/Timer (`ICTR`) functions, and the Counter/Timer (`CTR`) functions. These sets of functions perform a variety of timing I/O and counter operations such as event counting, period and frequency measurement, and single-pulse and pulse-train generation. See your hardware user manual to find out which operations are supported by your device. Table 3-8 shows the sets of functions according to the devices they support.

Note For TIO-based 45*XX* DSA devices, use only counters 0 and 1. Refer to the *NI-DAQ Function Reference Online Help* or the device user manual for counter pinouts.

Table 3-8. Devices Supported by the `GPCTR`, `ICTR`, and `CTR` Functions

Functional Set	E-Series, 622*X*, 660*X*, 671*X*, and DSA Devices	516 Devices, DAQCard-500/700, Lab and 1200 Devices, LPM Devices	PC-TIO-10
`GPCTR`	yes	no	no
`ICTR`	no	yes	no
`CTR`	no	no	yes

Counter/Timer Functions

The Counter/Timer (CTR) 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 the specified counter operation.
CTR_Simul_Op	Configures and simultaneously starts and stops multiple counters.
CTR_Square	Causes the specified counter to generate a continuous square wave output of specified duty cycle and frequency.

`CTR_State`	Returns the OUT logic level of the specified counter.
`CTR_Stop`	Suspends operation of the specified counter so that NI-DAQ can restart the counter operation.

Counter/Timer Operation for the CTR Functions

Figure 3-26 shows the 16-bit counters available on the PC-TIO-10.

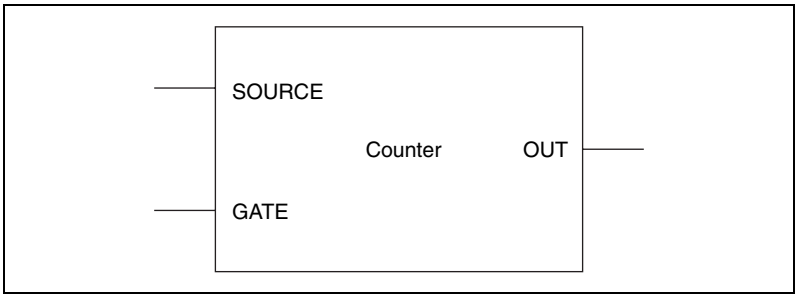


Figure 3-26. Counter Block Diagram

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

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

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

Note A 5 MHz internal timebase (200 ns resolution) is also available on SOURCE 5 for counters 1 to 5, and SOURCE 10 for counters 6 to 10 on the PC-TIO-10.

You can also 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 software-configure a counter 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 signal application at the GATE input and the start of the counter operation.

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

The counters generate two types of output signals during counter operation—TC pulse output and TC toggled output. A counter reaches TC when it counts up to 65,535 or down to 0 and rolls over. In many counter applications, the counter reloads from an internal register when it reaches TC. In TC pulse output mode, the counter generates a pulse during the cycle in which it reaches TC. In TC toggled output mode, the counter output changes state on the next source edge after reaching TC. In addition, you can configure the counters for positive logic output or negative (inverted) logic output. Figure 3-27 shows examples of the four types of output signals generated.

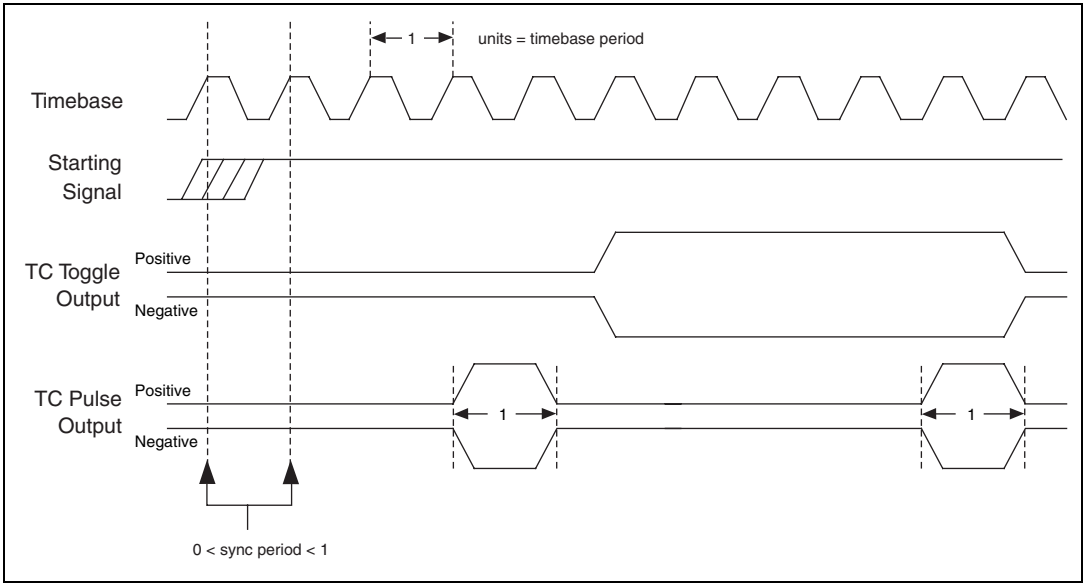


Figure 3-27. Counter Timing and Output Types

Figure 3-27 represents a counter generating a delayed pulse (see `CTR_Pulse` in the function reference) 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 3-27, represents either a software starting of the counter, for the no-gating case, or some sort of signal at the GATE input. The signal 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 3-27, the counter is configured to count the signal output changes state with respect to the rising edge of the timebase.

Programmable Frequency Output Operation

The PC-TIO-10 provides two 4-bit programmable frequency output signals. The signals are divided-down versions 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 Online Help* file for FOUT use and timing information.

Counter/Timer Application Tips

All NI-DAQ counter/timer functions can be broken down into two major categories—event-counting functions and pulse generation functions. On top of those functions, NI-DAQ has utility functions.

`CTR_EvCount` and `CTR_EvRead` are the two functions designed for event-counting. See Figure 3-28 for basic building blocks of event-counter applications. Also, read *Event-Counting Applications* later in this chapter for details.

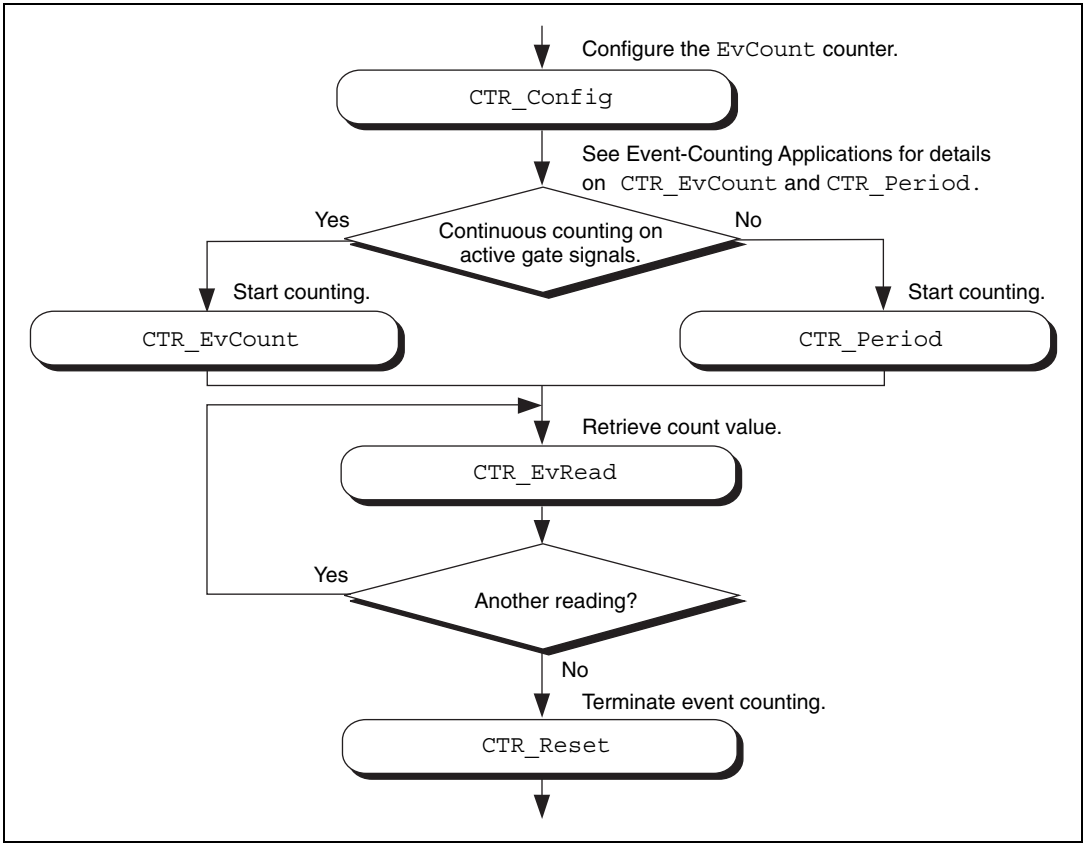


Figure 3-28. Event Counting

Another major category of counter functions is pulse generation. With the NI-DAQ counter functions, you can call `CTR_Pulse` to generate a pulse or `CTR_Square` to generate a train of pulses (a square wave). To generate a pulse or a square wave, see Figure 3-29 for details on the function flow. When `CTR_Square` is used with special gating (**gateMode** = 8), you can achieve gate-controlled pulse generation. When the gate input is high, NI-DAQ uses **period1** to generate the pulses. When the gate input is low, NI-DAQ uses **period2** to generate the pulses. If the output mode is TC Toggled, the result is two 50 percent duty square waves of different frequencies. If the output mode is TC Pulse, the result is two pulse trains of different frequencies.

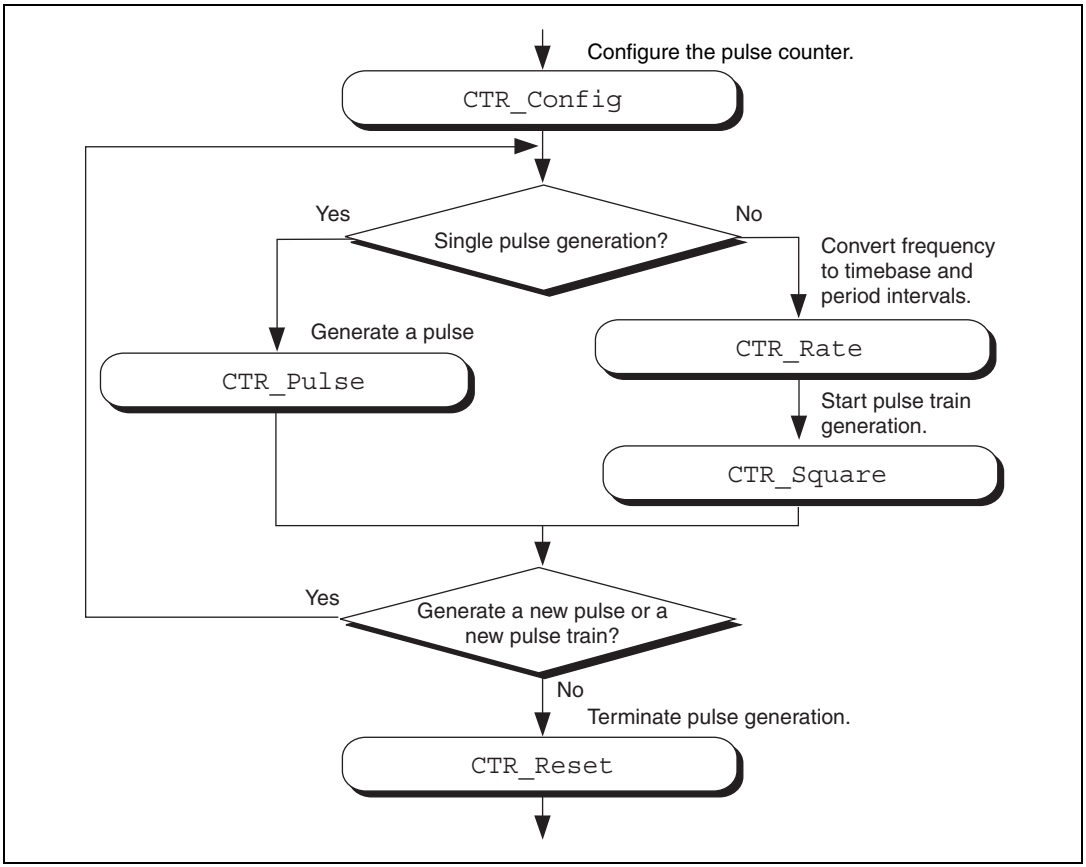


Figure 3-29. Pulse Generation

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` can only generate a square wave with a 50 percent duty cycle.

NI-DAQ has a number of utility functions that give you 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 3-30 on how to incorporate `CTR_Simul_Op` with other counter functions like `CTR_EvCount` and `CTR_Pulse`.

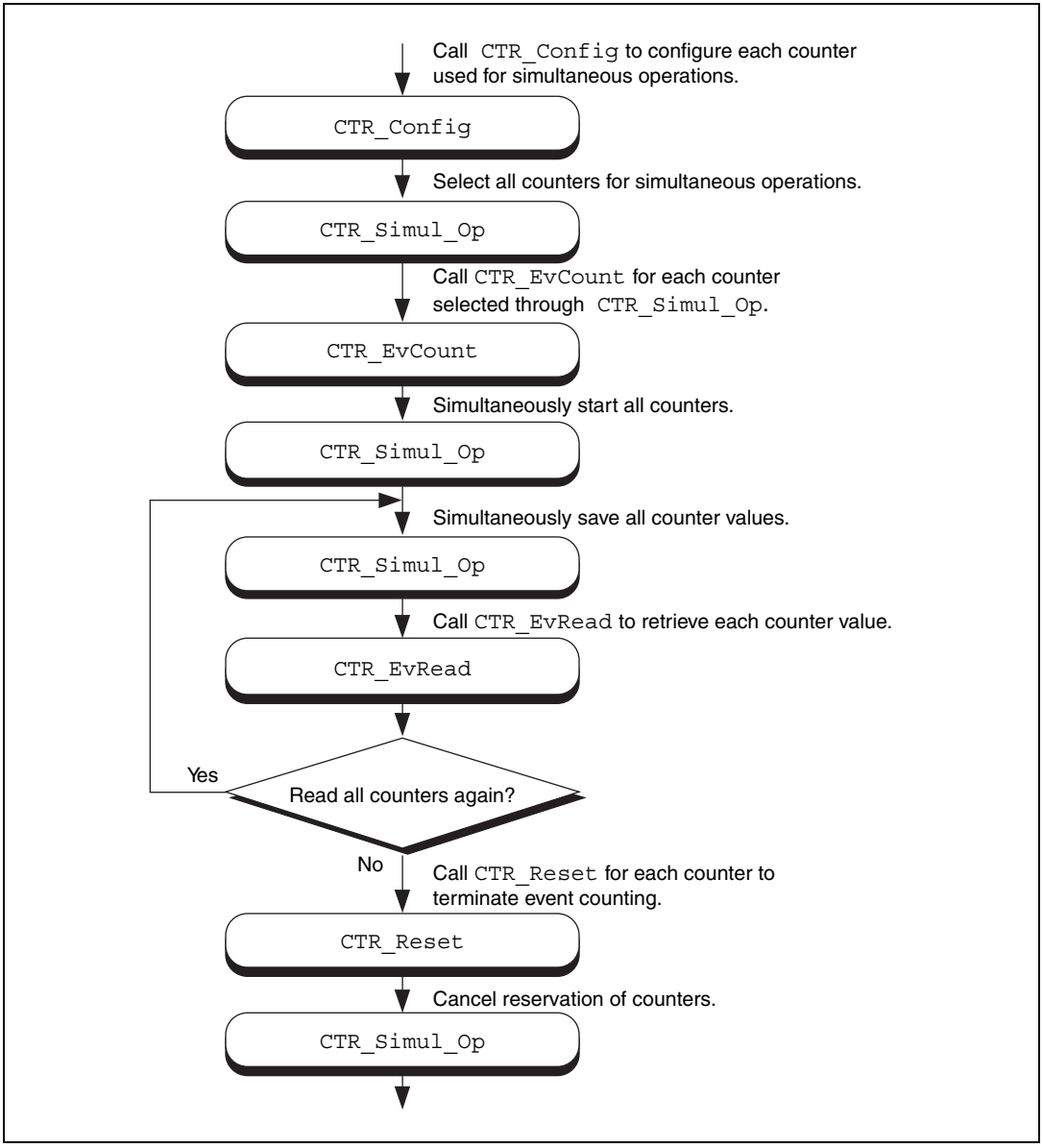


Figure 3-30. Simultaneous Counter Operation

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 so 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 6–10. NI-DAQ can count either low-to-high or high-to-low edges (this feature is selected by **edgeMode** in the CTR_Config function). In addition, you can use the various gating modes of CTR_Config to control counting. Figure 3-31 illustrates timer event counting.

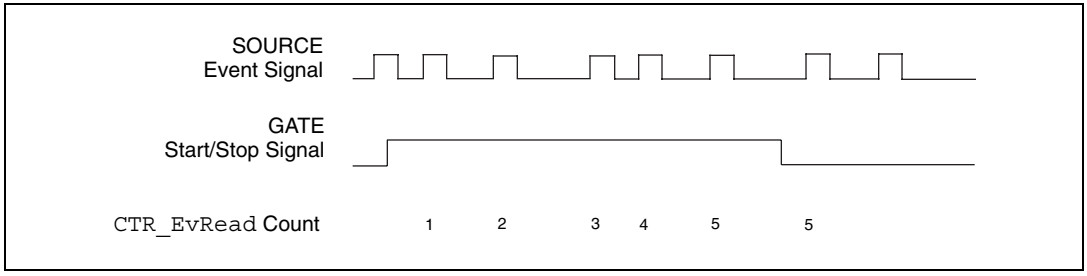


Figure 3-31. Timer Event Counting

For pulse-width measurement, configure a counter to count for the duration of a pulse. For this application, you can use any timebase, including an external clock connected to the counter SOURCE input. Use level gating modes for pulse-width measurements in which the pulse to be measured is connected to the counter GATE input. Pulse width is then equal to (event count) * (timebase period). Figure 3-32 shows a typical pulse-width measurement.

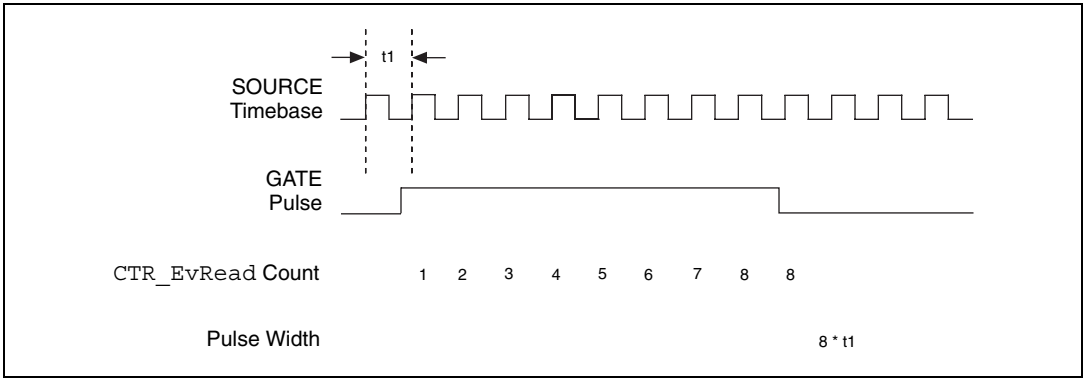


Figure 3-32. Pulse-Width Measurement

For time-lapse measurement, 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 `CTR_EvCount`. 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 signal edges 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). 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 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 description in the *NI-DAQ Function Reference Online Help* file for more information). Figure 3-33 illustrates a frequency measurement.

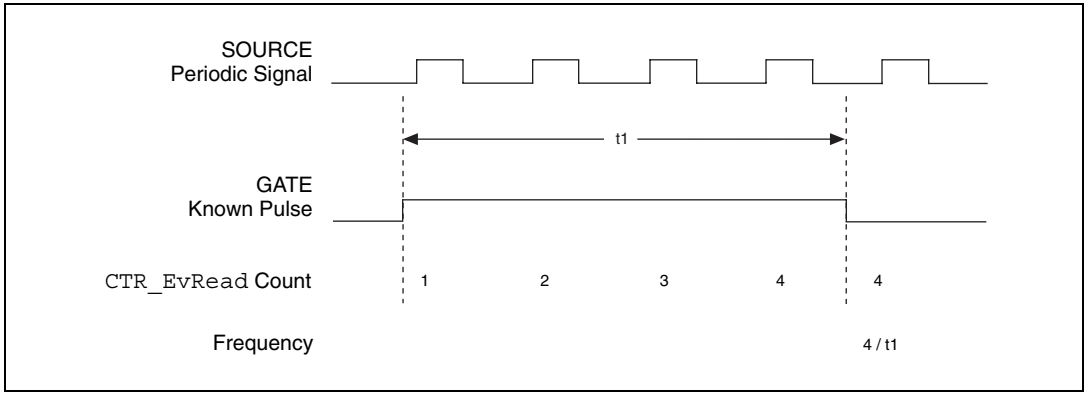


Figure 3-33. Frequency Measurement

For 16-bit resolution event counting and pulse width, time-lapse, or frequency measurement, you only need one counter. Select **cont** = 0 so that you are notified if the counter overflows (see the `CTR_EvRead` and `CTR_EvCount` function descriptions in the *NI-DAQ Function Reference Online Help* file). If **cont** = 1, event counting continues when the counter rolls over and no overflow condition is registered. **cont** = 1 is useful when more than one counter is concatenated for event counting.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) will perform 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 3-33) between two rising edges of the gate signal. With falling edge-triggered gating, a counter can measure the time interval between two falling edges of the gate signal. After you call `CTR_Config` and apply the signal being measured to the appropriate gate, you can call `CTR_Period` to initiate period measurement. The specified counter starts counting on the first gate edge and latches the counter value to the onboard Hold Register after the counter detects a second gate edge. After each period measurement, the counter reloads itself with a 0 and starts a new measurement. Figure 3-34 shows a continuous period measurement.

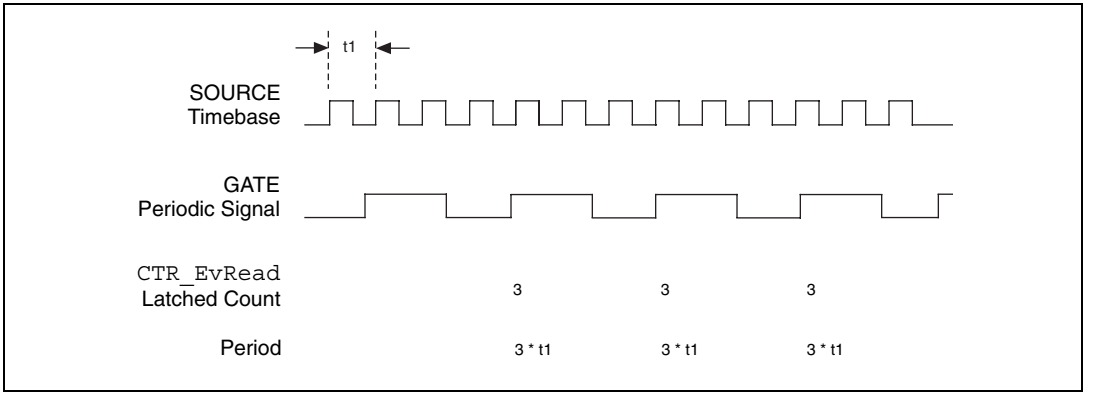


Figure 3-34. Continuous Period Measurement

While the measurement is occurring, call `CTR_EvRead` to retrieve the counter value saved in the Hold Register. The period is then equal to the value returned by `CTR_EvRead` * **timebase**.

If you choose an improper timebase frequency, `CTR_EvRead` retrieves a smaller count value. A small count indicates that the timebase frequency is either too low or too high compared to the gate signal. If the timebase frequency is too low, the counter can only count a few source edges. However, if the timebase frequency is too high, the counter counts too many source edges, causing counter overflow. In case of counter overflow, a small count (typically 1 or 2) is saved on the Hold Register, and the counter reloads itself with a zero and waits for a new gate trigger to make a new measurement.

For a pulse-width measurement, you 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.

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.

Refer to the *NI-DAQ Function Reference Online Help* file to determine which functions your device supports.

`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 3-35 shows the 16-bit counters available on the 516 devices, DAQCard-500/700, Lab and 1200 devices, and LPM devices.

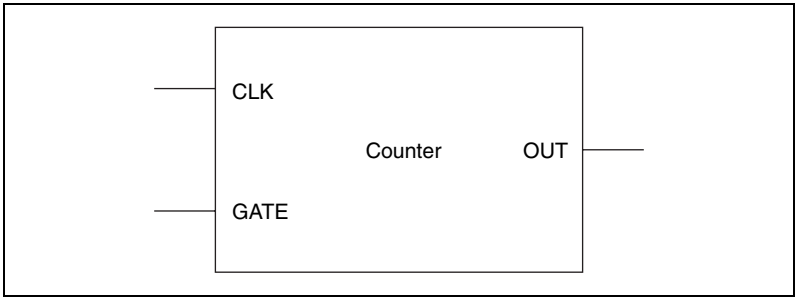


Figure 3-35. Interval Counter Block Diagram

Each counter has a CLK input, a GATE input, and an OUT output signal. Use a counter to count the falling edges of the signal applied to the CLK input. The counter GATE input gates counting operations. If your device uses an 8253 or MSM82C54 chip, refer to the data sheet in your device-specific manual to see how the GATE inputs affect the counting operation in different counting modes.

Interval Counter/Timer Application Tips

NI-DAQ interval counter functions 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 Online Help* file 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`.

General-Purpose Counter/Timer Functions

Use the General-Purpose Counter/Timer (GPCTR) functions with the E Series, 622*X*, 660*X*, 671*X*, NI-TIO based, and DSA devices. Refer to the GPCTR functions in the *NI-DAQ Function Reference Online Help* file for a detailed description of how to use the GPCTR functions for a variety of applications.

GPCTR_Change_Parameter	Customizes the counter operation to fit the requirements of your application by selecting a specific parameter setting.
GPCTR_Config_Buffer	Assigns the buffer that NI-DAQ uses for a buffered counter operation.
GPCTR_Control	Controls the operation of the general-purpose counter.
GPCTR_Read_Buffer	Transfers data from the previously assigned buffer during an asynchronous counter operation.
GPCTR_Set_Application	Selects the application for which you use the general-purpose counter. The function description in the *NI-DAQ Function Reference Online Help* file contains many application tips.
GPCTR_Watch	Monitors the state of the general-purpose counter and its operation.

General-Purpose Counter/Timer Application Tips

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. When using the GPCTR functions, follow the generic program flow as shown in Figure 3-36.

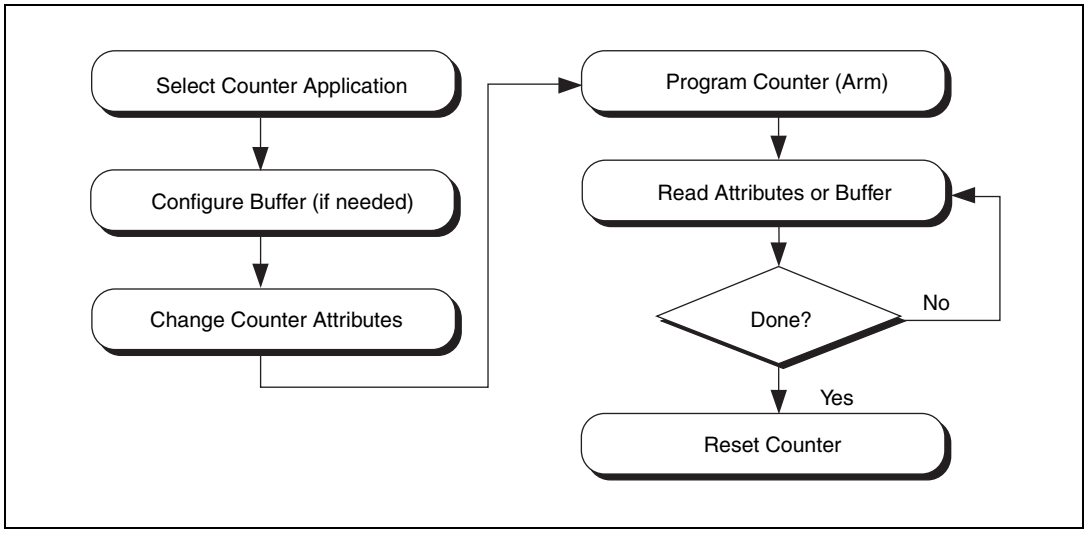


Figure 3-36. Generic Program Flow for All GPCTR Counter Applications

To select the type of application you want to use, (for example simple event counting, buffered event counting, period measurement, and so on) call the GPCTR_Set_Application function with the appropriate application parameter. If the application is buffered, configure a buffer for use during the acquisition using the GPCTR_Config_Buffer function. Next, change some of the counter attributes, depending on your type of application, by calling GPCTR_Change_Parameter. For example, set the counter source to the internal 100 kHz timebase; set the initial value of the counter to 0; or set the output mode of the counter to pulse mode. Arm the counter with the settings you made, by calling GPCTR_Control.

If you configure the counter to use a start trigger, the counter will not start counting until it receives the start trigger signal. Otherwise, the counter immediately begins counting. Check the status of the counters by using the GPCTR_Watch function. If you are doing a continuous buffered operation, read the buffer by calling GPCTR_Read_Buffer. When the operation has completed or you want to abort the operation, reset the counter by calling GPCTR_Control with appropriate control code.

For more about GPCTR functions, refer to the GPCTR_Set_Application function description in *NI-DAQ Function Reference Online Help* file.

Clocks or Time Counters

NI-TIO based devices have built-in clocks, which are specialized time counters that retrieve current time and timestamp one or more digital triggers. The number of clocks available depends on the number of NI-TIO chips on your device. Most devices have one clock per NI-TIO chip.

Like counters, clocks have gate signals that latch their current value, or time. They can latch a single time or multiple times in a buffer using interrupts or DMA. Unlike counters, clocks have additional hardware that eliminates drift by synchronizing the clocks to a PPS or IRIG-B stream (See Figure 3-37).

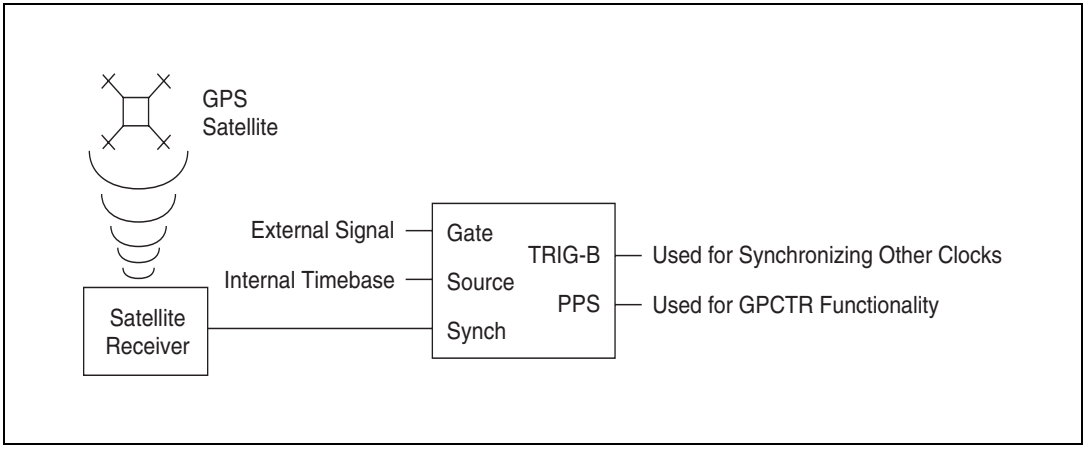


Figure 3-37. Real-time Clock Synchronizing to a GPS Receiver

Clock Resolution

NI-TIO based clocks must have a timebase, which keeps the clock ticking. The default timebase is the 20 MHz internal timebase. If your clock has a 20 MHz timebase, its resolution will be 50ns.

$$\frac{1}{20}\ \text{MHz} = 50\text{ns}$$

Clock Synchronization

Clocks can be synchronized using an external source. Currently, there are two supported types of synchronization—Pulse Per Second (*PPS*), and *IRIG* B.

Pulse Per Second

PPS, the simplest type of synchronization, is a very accurate 1 Hz signal. Depending on your satellite receiver, the accuracy of this signal can vary from 300 to 100 ns. The disadvantage of PPS is that no absolute timing information is encoded in the pulse, so you must set the initial time of the clock. After that, the PPS keeps the subsecond counters synchronized. The initial time for NI-TIO clocks is specified in seconds from 12:00 a.m., January 1 of the current year. You can think of this as *Julian* seconds if it is easier. Most GPS receivers produce a PPS stream.

You must allow for a window of error when using PPS. When you receive the time from the receiver, you must set the clock up within 1 second. If you wait any longer, your clock will be off by 1 or more seconds. The *SetUpClockUsingPPS* section shows pseudocode for ensuring that your clock has been set safely.

SetUpClockUsingPPS

```
Comment: Get the latest time from the RS-232 port of the
receiver. It has a Comment: resolution of one second.
    initialTime = Satellite Receiver Time
    lastTime = initialTime
Comment: This is for initialization.
    While (initialTime = lastTime)
    {
Comment: Get the latest time.
        lastTime = Satellite Receiver Time
    }
Comment: We now are at the beginning of the next second.
Comment: This will minimize the chance of taking too long
Comment: to program the clock.
GPCTR_Control(deviceNumber, ND_CLOCK_x, ND_RESET)
GPCTR_Set_Application(deviceNumber, ND_CLOCK_x,
    ND_SIMPLE_TIME_MSR)
```

```
    GPCTR_Change_Parameter(deviceNumber, ND_CLOCK_x,
        ND_SYNCHRONIZATION_METHOD, ND_PULSE_PER_SECOND)
    GPCTR_Change_Parameter(deviceNumber, ND_CLOCK_x,
        ND_SECONDS,lastTime)
    GPCTR_Control(deviceNumber, ND_CLOCK_x, ND_PROGRAM)
Comment: We have programmed the clock. Now we need to
Comment: make sure that the Comment: programming was
Comment: really successful.
    timeAfterProgramming = Satellite Receiver Time
    if (timeAfterProgramming = lastTime)
    {
Comment: We were successful in programming the clock in
Comment: time!
    }
    else
    {
Comment: We didn't make it in time. Try again…
    }
}
```

IRIG

The Inter Range Instrumentation Group (IRIG) defines a series of time-code protocols, which transmit a series of data frames that provide timing information in the binary data of the frame as well as at the start of frame time. The clock synchronization uses a subset of the IRIG-B protocol for synchronization. The IRIG-B frame consists of 100 bits, each of which is pulse-width modulated for 10 ms. The resulting data stream repeats once per second with the beginning of the frame marking the 1 second epoch (See Figure 3-38).

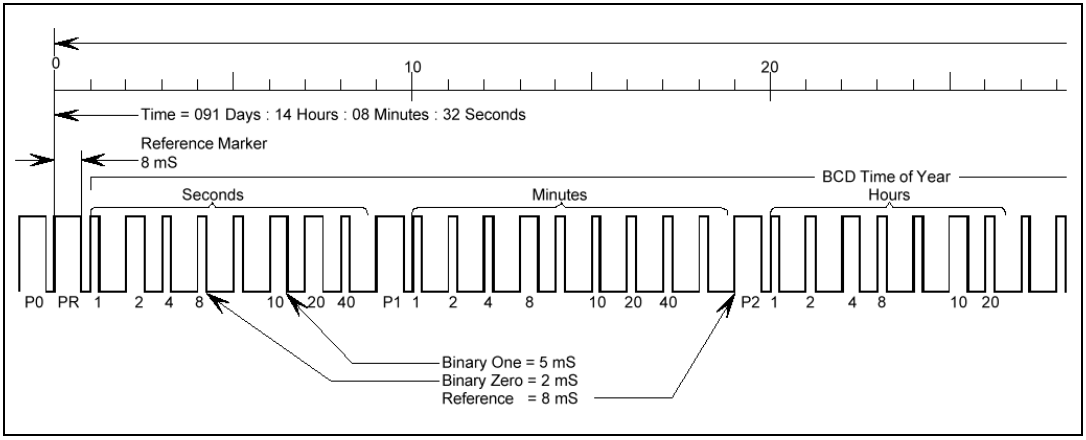


Figure 3-38. IRIG-B Transmission Frame

Clock Accuracy

Clock accuracy depends on the resolution of the clock source and the accuracy of the synchronization pulse. The source of your IRIG-B stream determines the accuracy of the synchronization pulse. You can calculate the clock accuracy by adding the resolution of the clock source and the accuracy of the synchronization pulse, both in nanoseconds. For example, if you have a 50ns resolution and a 100ns synchronization pulse accuracy, your clock accuracy is as follows:

$$50\text{ns} + 100\text{ns} = 150\text{ns}$$

Example Clock in a Measurement System

The block diagram in Figure 3-39 illustrates a system in which all the clock capabilities of the TIO can be exploited. The chassis has its TIO board, 6602, synchronized to a GPS, but the other modules are not clock-capable. Each of the other modules is programmed to assert a PXI trigger when it receives an event. The TIO board is set up to time stamp the triggers of the other modules as well as to save the clock value on the first time stamp. With the system configured, you can determine the precise relationship of all triggers throughout the system to correlate the data. If you clone the system, you can correlate all the data from the individual systems as a whole.

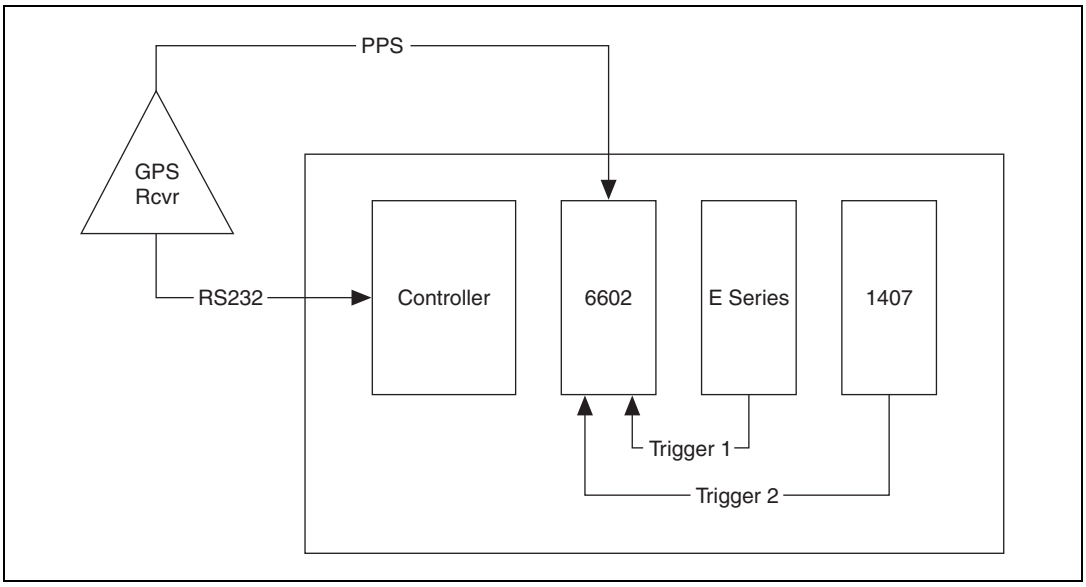


Figure 3-39. Example Clock in a Measurement System

Sample Use Cases

The following cases are examples of applications that can use clocks.

Use Case #1

The user records the time of *n* events with software timing precision.

A user wants to record the time of an external event—for example, when a key is pressed on a keyboard, when a data packet is received by a CAN controller, or when a temperature is read from an AI channel.

The software program flow is as follows:

1. Initialize the clock for this application.
   - Specify the synchronization signal type (none).
   - Specify the initial DHMS value (the default value of DHMS is undefined).
2. Read the clock value (days, hours, minutes, seconds) when the external event happens to determine the time of the event.
3. At the end of the operation, free up any hardware and software resources. After powering up, the clock will run until the power to the TIO ASIC is turned off.

Use Case #2

The user records the time of a single event on two physically distributed data acquisition systems with hardware timing precision.

A user wants to record how a lightning strike at a power grid in city A travels to a grid station in city B. The lightning sensors in both cities produce a TTL pulse upon receiving the lightning bolt in city A.

Another user wants to correlate buffered analog input measurements across distributed data acquisition systems. He connects a GPS receiver to an NI-TIO device on each system and connects the scan clock to the gate input of the clock. The first active edge on the scan clock on each system will record the global time that sample was taken. The user can use the sample rate on each system to calculate the absolute time each sample was taken, assuming sample rates remain constant throughout the experiment.

The software program flow is as follows:

1. Initialize the clock for this application.
   - Specify the synchronization signal source (PFI line, RTSI line).
   - Specify the synchronization method.
     - If source is PPS, read the current time from the GPS receiver via a serial port and set it as the initial DHMS value of the clock, or choose any arbitrary value. Initialization will take about 2 seconds.
     - If source is IRIG-B, simply wait for 2 seconds after programming the hardware. The clock will automatically synchronize to the GPS signal.
   - Specify gate signal source and polarity.
   - Clock value automatically latches upon receiving the first pulse on its gate input.
2. Poll the armed attribute of the clock until it is disarmed (when the external event happens). The clock is always armed in hardware, but the software abstraction of the clock will be unarmed after it latches the first point.
3. Read the clock value (days, hours, minutes, seconds) to determine the time of the external event.
4. At the end of the operation, free up any hardware and software resources. After powering up, the clock will run until the power to the TIO ASIC is turned off.

Use Case #3

The user generates a single trigger pulse event on two physically distributed data acquisition systems.

A user wants to generate a sine wave at location A and make buffered analog input measurements at location B at precisely the same time. The sine wave generated at location A affects the measurements made at location B.

The software program flow is as follows:

1. Initialize the clock for this application.
   - Specify the synchronization signal source (PFI line, RTSI line).
   - Specify the synchronization method.
     - If source is PPS, read the current time from the GPS receiver via a serial port and set it as the initial DHMS value of the clock, or choose any arbitrary value. Initialization will take about 2 seconds. For more information on initializing the clock with PPS, refer to the *SetUpClockUsingPPS* section.
     - If source is IRIG-B, simply wait for 2 seconds after programming the hardware. The clock will automatically synchronize to the GPS signal.
   - Route the PPS output of the receiver to one of the counter source lines for the TIO counter to receive.
2. Set up the counter.
   - Configure one of the counters for single pulse generation.
     - Application type = single pulse generation
     - Source = source line carrying the PPS signal
   - Specify the pulse delay.
     - Translate the actual trigger time into the number of seconds since the beginning of the current year (*x* seconds).
     - Read the clock value (*y* seconds).
     - Specify pulse delay (*x* - *y*).
   - Specify the appropriate trigger pulse width.
   - Arm the counter.
   - Read the clock value.
     - If the clock value is still *y* seconds, you are done.
     - If the clock value is more than *y* seconds, reset the counter and return to the beginning of step 2.

- Configure analog input (analog output) operation to trigger upon receiving a pulse on the counter output.
- Counter produces a pulse when the clock value reaches the desired time interval.

3. Poll the armed attribute of the counter until it is disarmed, when the counter has generated the trigger pulse.
4. At the end of the operation, free up any hardware and software resources. After powering up, the clock will run until the power to the TIO ASIC is turned off.

RTSI Bus Trigger Functions

The Real-Time System Integration (RTSI) Bus Trigger functions connect and disconnect signals over the RTSI bus trigger lines.

Refer to the *NI-DAQ Function Reference Online Help* file to determine which functions your device supports.

`RTSI_Clear`	Disconnects all RTSI bus trigger lines from signals on the specified device.
`RTSI_Clock`	Connects or disconnects the system clock from the RTSI bus.
`RTSI_Conn`	Connects a device signal to the specified RTSI bus trigger line.
`RTSI_DisConn`	Disconnects a device signal from the specified RTSI bus trigger line.
`Select_Signal`	Connects or disconnects a device signal to a RTSI bus trigger line.

Refer to the *NI-DAQ Function Reference Online Help* file to determine if your device supports RTSI.

RTSI Bus

The RTSI bus is implemented via a 34-pin ribbon cable connector on the AT, NEC, PCI E Series, PCI-DSA, and 671*X* devices. The RTSI bus is implemented on VXI-DAQ devices using the VXIbus trigger lines. On PXI DAQ and DSA devices, the RTSI bus is implemented using PXI trigger bus lines. On 1394 E Series DAQ devices, the RTSI bus is implemented with a 15-pin mini-*DSUB* connector. The RTSI bus has 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. 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.

E Series, DSA, 660*X*, and 671*X* RTSI Connections

For information regarding signals on the E Series, DSA, 660*X*, and 671*X* devices that you can connect to the RTSI bus, refer to the `Select_Signal` function description in the *NI-DAQ Function Reference Online Help* file.

Note If you have a PXI-DSA board in slot 2 (star trigger controller slot), do not drive any signals on RTSI 6 from other modules in the chassis. You can use other RTSI lines.

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 3-9 shows these signals.

Table 3-9. AT-AO-6/10 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 device. 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 your device user manual.

DIO-32F RTSI Connections

The DIO-32F contains four signals that you can connect to the RTSI bus trigger lines. Table 3-10 shows these signals.

Table 3-10. DIO-32F RTSI Bus Signals

Signal Name	Signal Direction	Signal Code
REQ1	Receiver	0
REQ2	Receiver	1
ACK1	Source	2
ACK2	Source	3

The signals REQ1 and REQ2 are request signals received from the I/O connector. An external device drives these signals during handshaking. ACK1 and ACK2 are supplied for handshaking with the DIO-32F over the RTSI bus. For more information about the DIO-32F signals, see the *AT-DIO-32F User Manual*.

653*X* RTSI Connections

The 653*X* devices (except for the DAQCard-653*X*) contain eight signals that you can connect to the RTSI bus trigger lines. Table 3-11 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 RTSI receiver when you use that signal as a device output. For example, do not configure the 653*X* 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 3-11. 653*X* 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 (STARTTRIG2)	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

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 653*X* signals, refer to the *DIO* 653*X User Manual*. Find additional explanations of the ACK1, ACK2, STOPTRIG1, and STOPTRIG2 signals in the `DIG_Trigger_Config` function in the *NI-DAQ Function Reference Online Help* file.

RTSI Bus Application Tips

This section gives a basic explanation of how to construct an application that uses RTSI bus NI-DAQ functions. Use the flowcharts as 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. Execute the work of the application.
3. Disconnect the signals from the RTSI bus. Figure 3-40 illustrates the normal order of RTSI function calls.

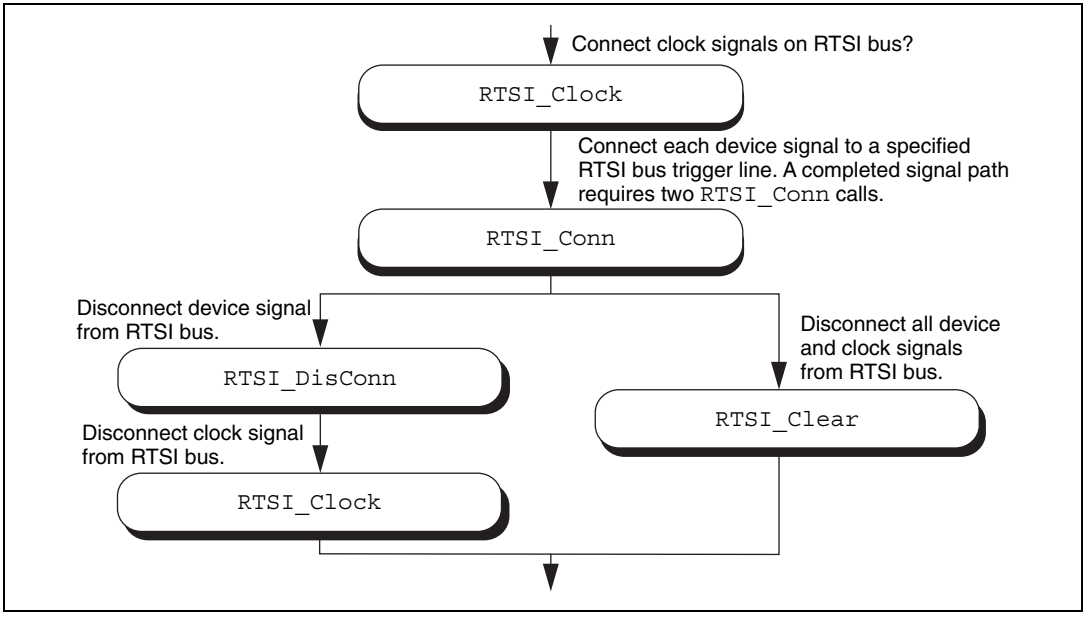


Figure 3-40. Basic RTSI Application Calls

Call RTSI_Clock/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 finish with the RTSI bus, disconnect the device from the bus. To do this, call RTSI_DisConn/RTSI_Clock for each connection made. Alternatively, call RTSI_Clear to sever all connections from your device to the RTSI bus.

SCXI Functions

Refer to the *NI-DAQ Function Reference Online Help* file to determine which functions your device supports.

`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`	Provides a single call calibration for the SCXI-1112, SCXI-1125, SCXI-1520, and SCXI-1540 modules. With the SCXI-1112 and SCXI-1125, calling

	this function calibrates individual channels. However, with the SCXI-1540, calling this function calibrates every channel. You also use this function to update the actual onboard reference voltage value on the SCXI-1125 EEPROM and to copy calibration constants to the module's default EEPROM load area from another EEPROM area or from NI-DAQ memory.
`SCXI_Calibrate_Setup`	Grounds the amplifier inputs of an SCXI-1100, SCXI-1101, SCXI-1122, or SCXI-1141, SCXI-1142, or SCXI-1143 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. This function supports shunt calibration for the SCXI-1122 module or the SCXI-1121 module with the SCXI-1321 terminal block. It also supports shunt calibration for the SCXI-1520 module with the SCXI-1314 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_Connection`	Sets the connection type parameter to a specified type on a given channel or all channels on the SCXI-1520 and SCXI-1540 modules. This function also allows programmatic control of external synchronization.
`SCXI_Configure_Filter`	Sets the specified channel to the assigned filter setting on any SCXI module with programmable filter settings.

`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, with the SCXI-1126 to determine if the module has settled after changing any of its programmable functions (ranges, filter settings, threshold, or hysteresis), or with the SCXI-1102/B/C to determine if the module has settled after changing gains.
`SCXI_Load_Config`	Loads the SCXI chassis configuration information that you established in the Measurement & Automation Explorer. 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 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 can also use SCXI_Reset to reset the Slot 0 scanning circuitry or to reset the entire chassis.
SCXI_Scale	Scales an array of binary data acquired from an SCXI channel to voltage.
SCXI_SCAN_Setup	Sets up the SCXI chassis for a multiplexed scanning data acquisition operation that the assigned DAQ device will perform. The function downloads a module scan list to Slot 0 that determines the sequence of scanned modules and how many channels on each module are scanned. This function can program each module with its given start channel, as well as resolve any contention on the SCXIbus.
SCXI_Set_Config	Changes the configuration of the SCXI chassis that you established in Measurement & Automation Explorer. Sets the software states of the chassis and modules specified to their default states. Does not change the SCXI chassis or module hardware states.
SCXI_Set_Excitation	Sets a specified excitation parameter to a supplied value on a given channel or all channels on the SCXI-1520, SCXI-1530, SCXI-1531, and SCXI-1540 modules.

`SCXI_Set_Gain`	Sets the specified channel to the given gain or range setting on any SCXI module that works with programmable gain or range settings.
`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_Set_Threshold`	Used to set the high and low threshold values for the SCXI-1126 frequency-to-voltage 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-and-hold modules track/hold state that you set up for a single-channel operation.
`SCXI_Track_Hold_Setup`	Establishes the track/hold behavior of a track-and-hold module and sets up the module for either a single-channel or an interval-scanning operation.

SCXI Application Tips

There are three categories of SCXI applications—analog input applications, analog output applications, and digital applications.

Figure 3-41 shows the basic structure of an SCXI application.

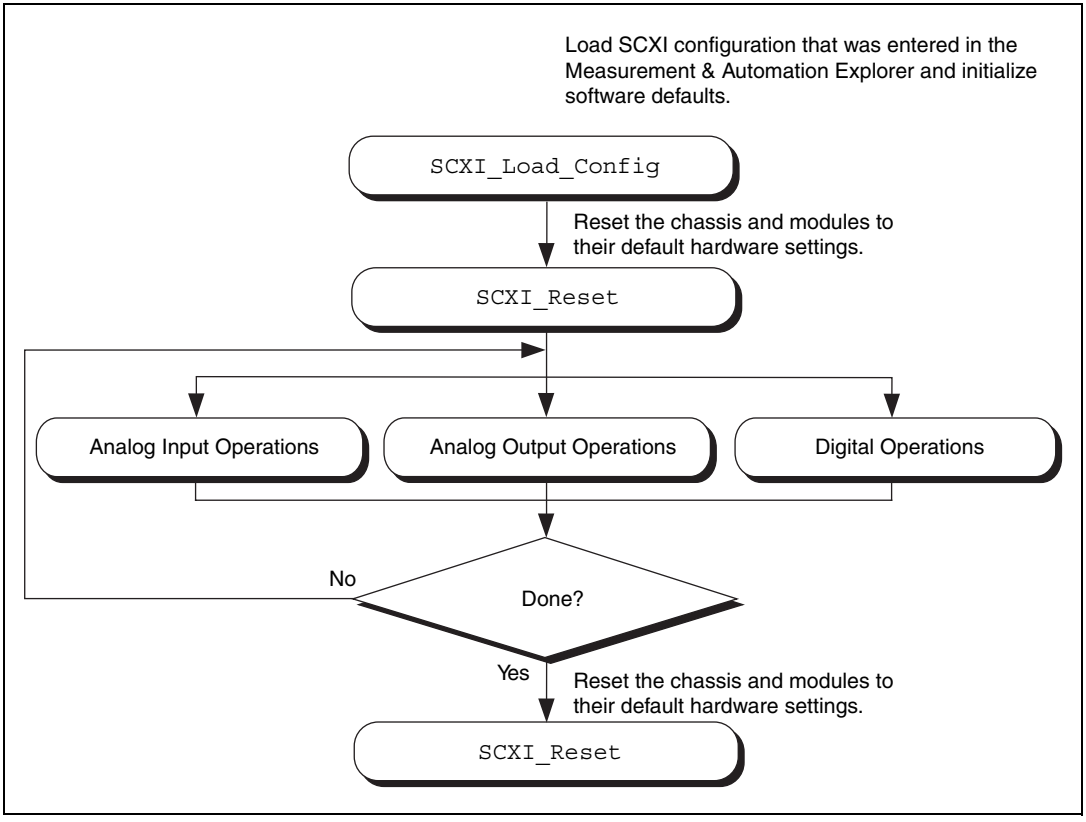


Figure 3-41. General SCXIbus Application

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 3-41. Please refer to the function descriptions in the *NI-DAQ Function Reference Online Help* file for detailed information about each function used in the flowcharts.

You can divide the SCXI analog input applications 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.

After you have set up the SCXI system, single-channel applications use the AI or the DAQ class of functions described earlier in this chapter to acquire the input data. 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 selected SCXI channel; this specific type of single-channel application is called *software scanning*.

After you have set up the SCXI system, channel-scanning applications use the SCAN and Lab_ISCAN classes of functions described earlier in this chapter to acquire the input data.

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. If you are using a PXI DAQ device with an internal connection to the PXI-1010 or PXI-1011 SCXIbus, then ND_PXI_SC is the **channel** parameter. 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 3-42 shows the function call sequence of a single-channel or software-scanning application using an SCXI-1100, SCXI-1101, SCXI-1102/B/C, VXI-SC-1102/B/C, SCXI-1104/C, SCXI-1112, SCXI-1120/D, SCXI-1121, SCXI-1122, SCXI-1125, SCXI-1126, SCXI-1141, SCXI-1142, SCXI-1143, SCXI-1520, SCXI-1530, SCXI-1531, or SCXI-1540 module operating in multiplexed mode.

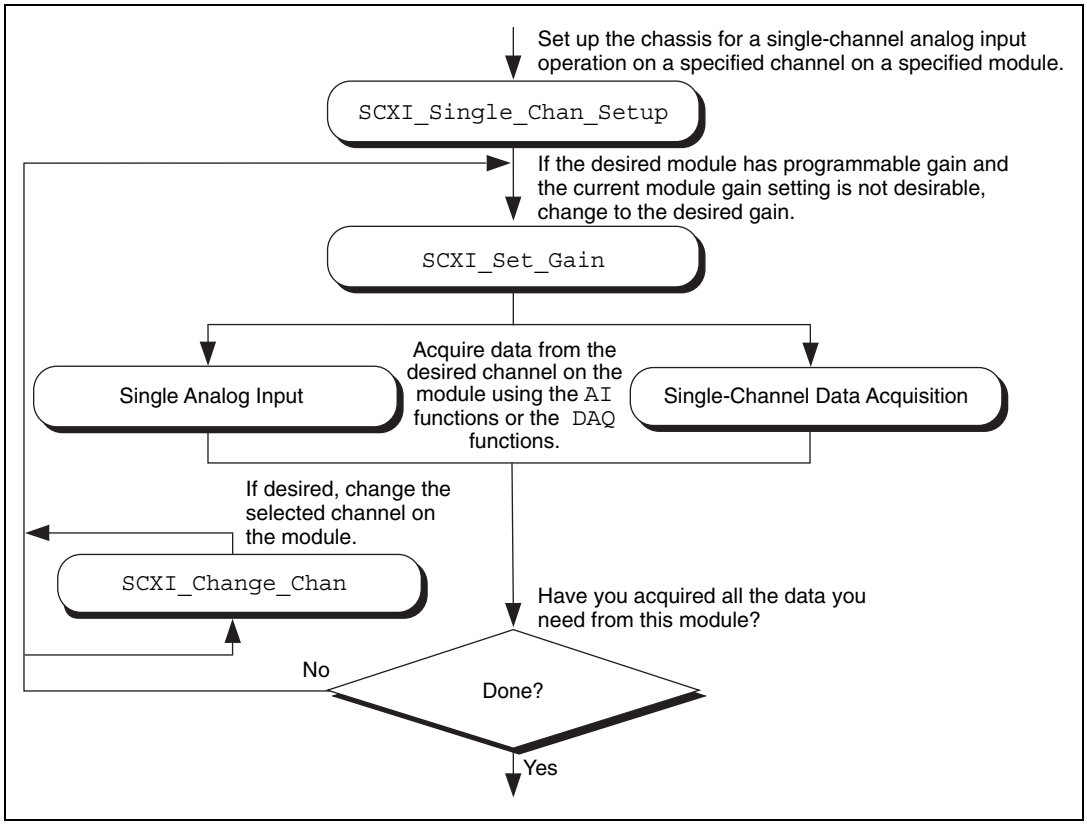


Figure 3-42. Single-Channel or Software-Scanning Operation in Multiplexed Mode

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 present on the SCXIbus to the DAQ device.

The SCXI_Set_Gain function changes the gain or range of the SCXI-1100, SCXI-1102/B/C, VXI-SC-1102/B/C, SCXI-1122, SCXI-1125, SCXI-1126, SCXI-1141, SCXI-1142, SCXI-1143, SCXI-1520, SCXI-1530, SCXI-1531, or SCXI-1540 module. The module maintains this gain or range setting until you call the function again to change it. You can also do any other module-specific programming at this point, such as SCXI_Configure_Filter or SCXI_Set_Input_Mode.

To achieve software scanning, select a different channel on the module using the SCXI_Change_Chan function after acquiring data from the channel you want with the AI or DAQ functions. If you want a channel on a different module, call the SCXI_Single_Chan_Setup function again to enable the appropriate module outputs and manage the SCXIbus signal routing.

Figure 3-43 shows the function call sequence of a single channel or software-scanning application using a Simultaneous Sample and Hold (SSH) module in multiplexed mode.

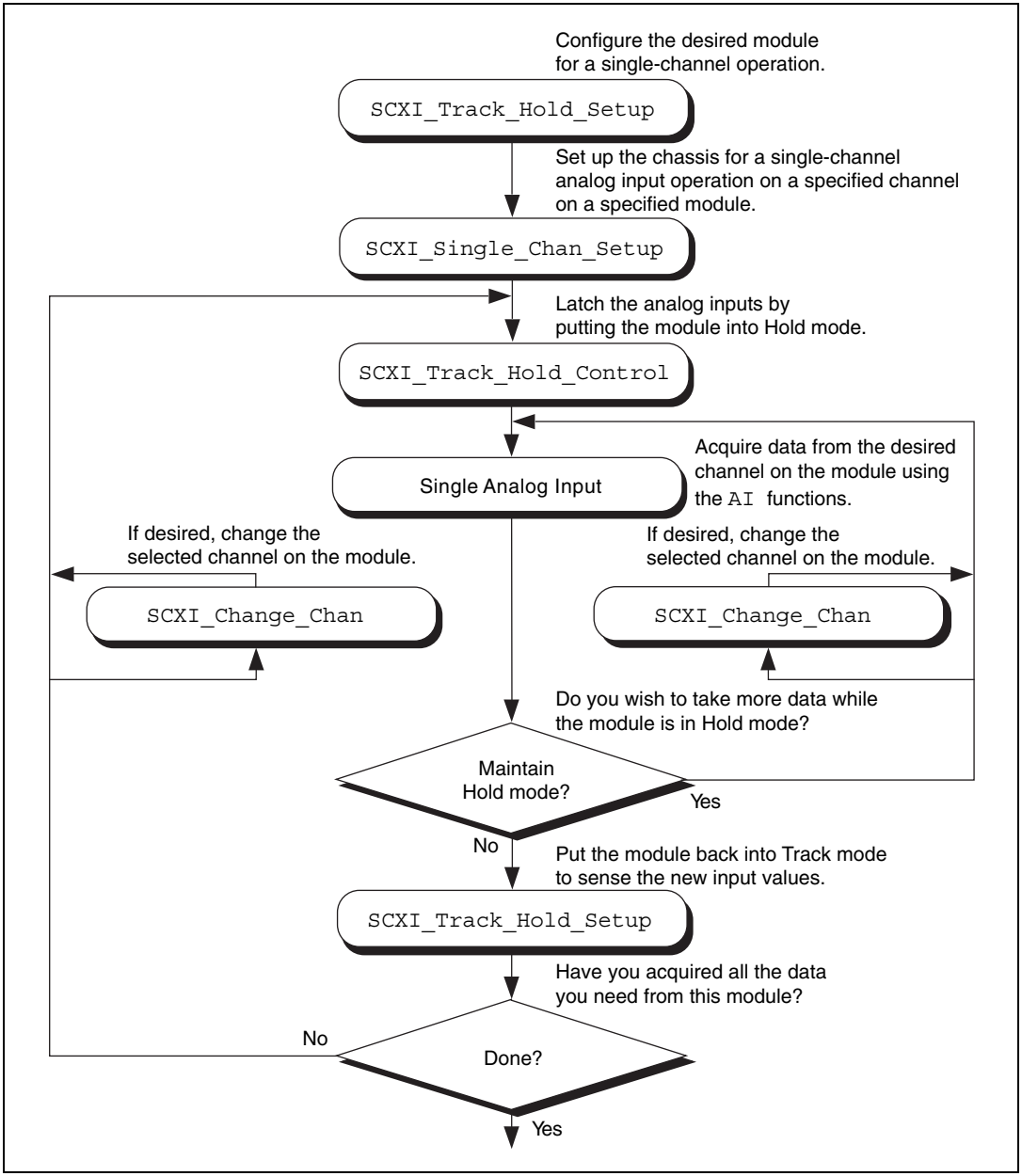


Figure 3-43. Single-Channel or Software-Scanning Operation Using a Simultaneous Sample and Hold Module in Multiplexed Mode

Notice the similarities between Figure 3-43 and Figure 3-44, 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 appropriately route the output to the DAQ device or SCXI-1200 module. 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 3-43 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 track-and-hold modules are to be scanned, use interval scanning; if you are using a plug-in DAQ device, the module directly connected to the DAQ device must be one of the SSH modules.

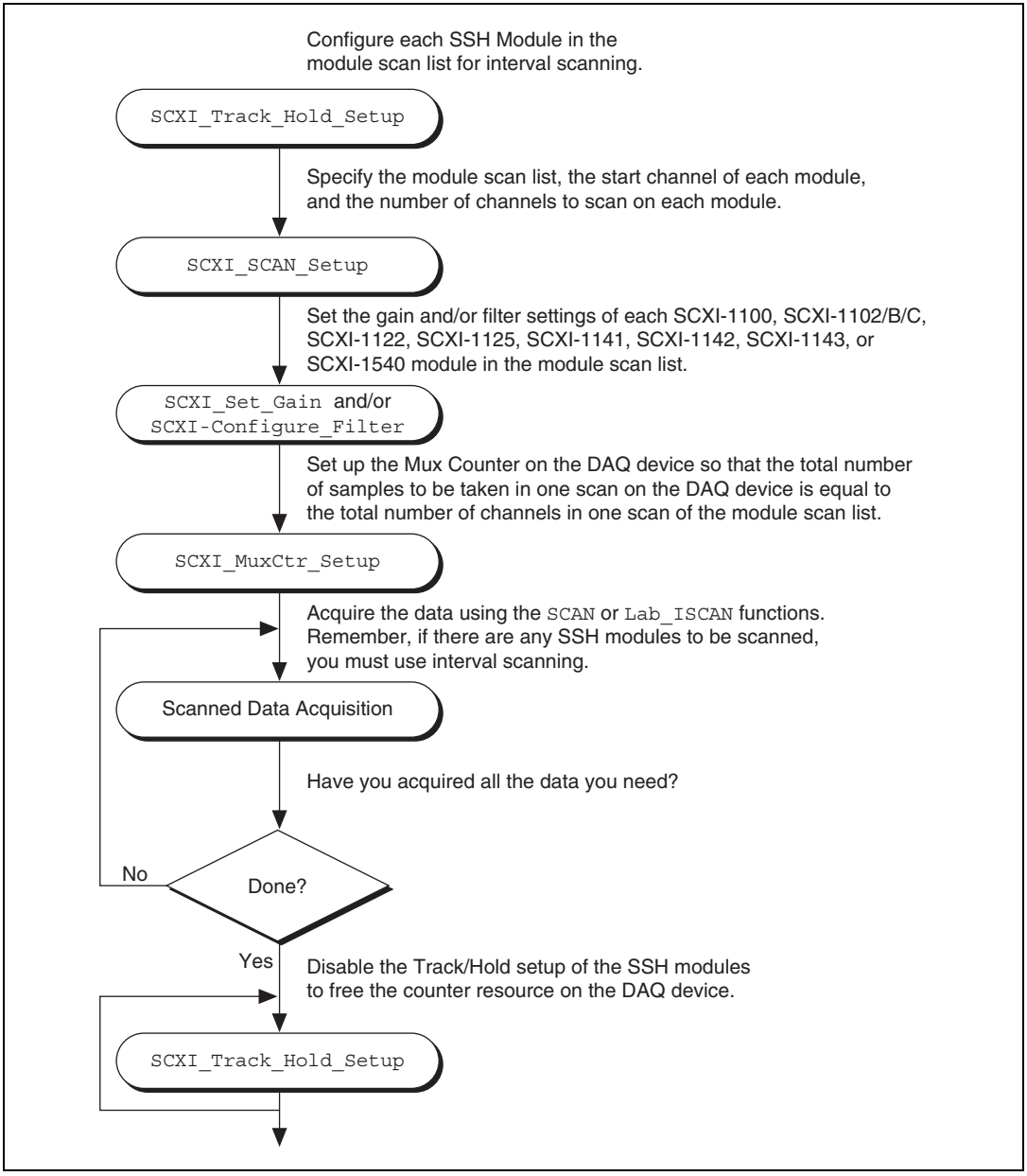


Figure 3-44. Channel-Scanning Operation Using Modules in Multiplexed Mode

If any of the modules to be scanned are SSH modules, you must establish the Track/Hold setup of each one. To synchronize multiple SSH 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 SSH module 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 the *NI-DAQ Function Reference Online Help* file, 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/B/C, VXI-SC-1102/B/C, SCXI-1122, SCXI-1125, SCXI-1126, SCXI-1141, SCXI-1142, SCXI-1143, SCXI-1520, SCXI-1530, SCXI-1531, or SCXI-1540 modules, you can use `SCXI_Set_Gain` to change the gain or range setting on each module. You also can use the `SCXI_Configure_Filter` function for the SCXI-1122, SCXI-1125, SCXI-1126, SCXI-1141, SCXI-1142, and SCXI-1143, SCXI-1530, SCXI-1531, 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 Online Help* file.

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/D, SCXI-1121, SCXI-1125, SCXI-1126, SCXI-1141, SCXI-1142, and SCXI-1143 modules in parallel mode, you need no further SCXI function calls beyond those shown in Figure 3-43 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/D, SCXI-1121, SCXI-1125, SCXI-1126, SCXI-1141, SCXI-1142, or SCXI-1143 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 module input.

In many of the data acquisition function descriptions in the *NI-DAQ Function Reference Online Help* file, 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 Measurement & Automation Explorer. You cannot use the SCXI-1200 to read channels from other analog input modules that are configured for parallel mode.

The SCXI-1100, SCXI-1101, SCXI-1102/B/C, VXI-SC-1102/B/C, SCXI-1104/C, SCXI-1112, SCXI-1122, and SCXI-1540 operate in multiplexed mode only.

The SCXI-1140, SCXI-1520, SCXI-1530, and SCXI-1531 modules require 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 3-45 shows the function call sequence of a single-channel (or software-scanning) operation using these modules in parallel mode.

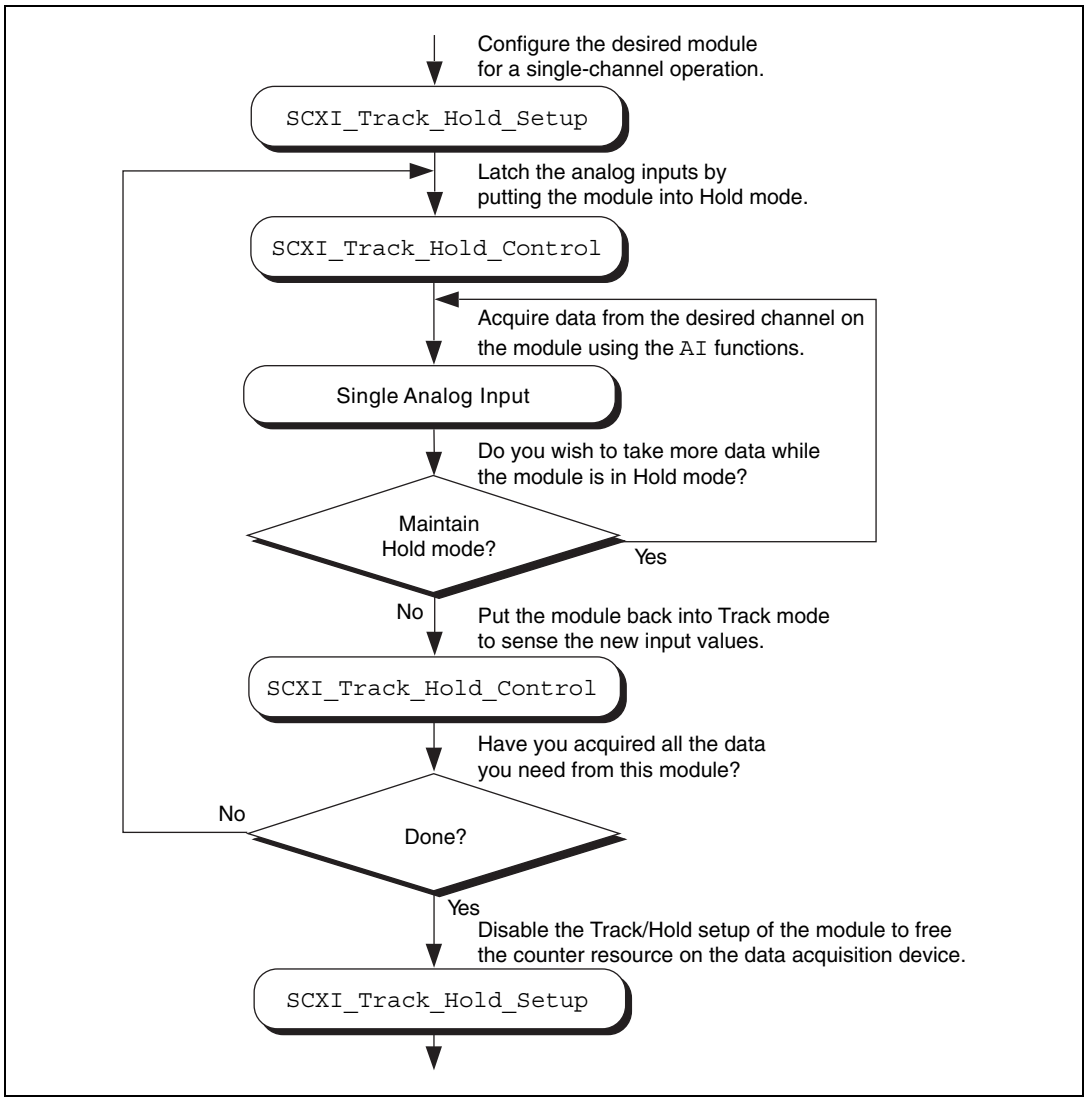


Figure 3-45. Single-Channel or Software-Scanning Operation Using an SSH Module in Parallel Mode

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 sampled voltages. 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. Also be aware of the SSH Track/Hold timing requirements described in the *SCXI-1530/1531* section of the *SCXI Hardware* chapter in the *DAQ Hardware Overview Guide*.

Figure 3-46 shows the function call sequence of a channel-scanning application using an SSH module in parallel mode.

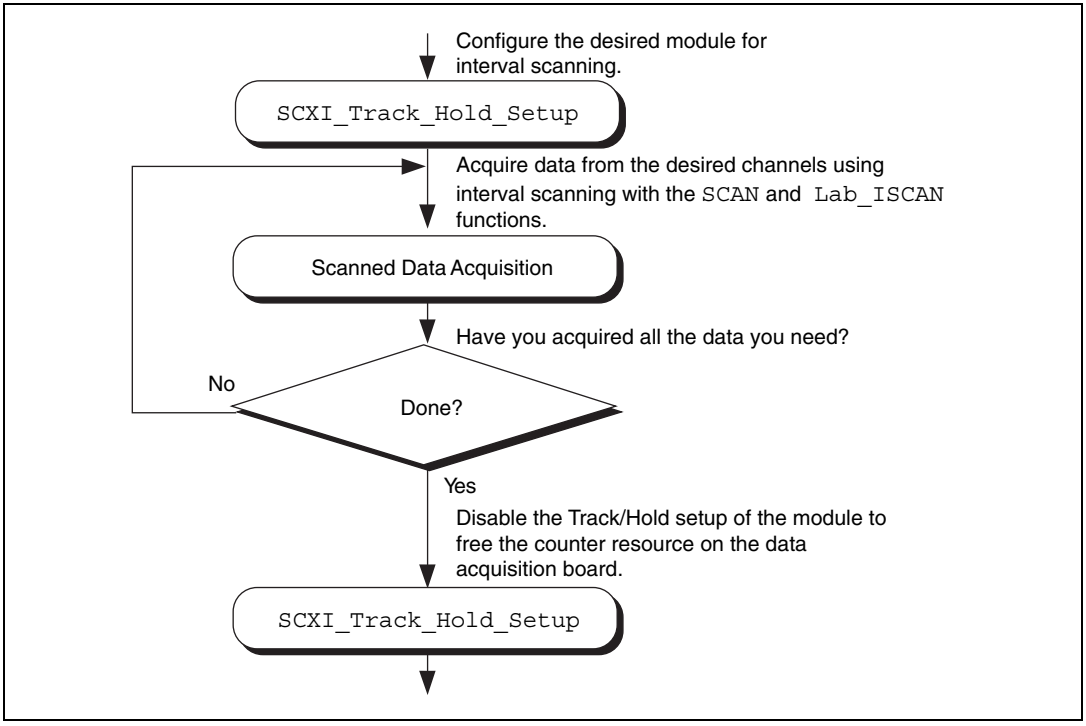


Figure 3-46. Channel-Scanning Operation Using an SSH Module in Parallel Mode

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 tan SSH 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 3-12 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 PCI-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 200 kS/s (such as the PCI-6032E), you should add 1 µs to the settling time of your DAQ device. The maximum acquisition rate for the PCI-6032E would be 1/(10 µs + 1 µs) = 90.9 kS/s.

If you are using a DAQ device faster than 200 kS/s 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 3-12. Maximum SCXI Module Settling Times

SCXI Module	Gain	Settling Time Using up to 12-bit, 200 kS/s[1] Device	Settling Time Using AT-MIO-16E-2[2]	Settling Time Using AT-MIO-16XE-10 (±0.006% Accuracy)[3]	Settling Time Using AT-MIO-16XE-10 (±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/B/C, VXI-SC-1102/B/C, SCXI-1104/C, SCXI-1112	all gains	7 µs	3 µs	10 µs	—
SCXI-1120, SCXI-1120D, SCXI-1125	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-1126	all gains or ranges	7 µs	3 µs	10 µs	20 µs
SCXI-1140	all gains	7 µs	3 µs	10 µs	20 µs
SCXI-1141, SCXI-1142, SCXI-1143, SCXI-1520, SCXI-1540	all gains	7 µs	3 µs	10 µs	20 µs

[1] Includes effects of a 12-bit, 200 kS/s device with 1 m SCXI cable assembly.

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

[3] Includes effects of AT-MIO-16XE-10 with 1 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 your SCXI user manual.

Table 3-13. 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 SCXI-1200 module acquisition rate 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 the *Specifications* appendix in your SCXI module user manual for details.

Note The SCXI-1122 uses relays to switch the input channels; the module requires 10 ms to settle when the relays 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 3-45 instead of the channel-scanning method shown in Figure 3-46.

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 simple. Call the `SCXI_AO_Write` function to write the voltages you want to the module DAC channels. You can use the `SCXI_Get_Status` function to determine when the DAC channels have settled to their final analog output voltages.

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/HV 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/R in parallel mode, `SCXI_Get_State` reads the hardware states.

If you are using the SCXI-1162/HV or SCXI-1163/R in parallel mode, you can use the SCXI functions as described above, or you can call the `DIG_In_Prt` and `DIG_Out_Prt` functions using the correct DAQ device port numbers that correspond to the SCXI module channels. The respective chapters on the DIO-96, DIO-24, AT-MIO-16DE1-10, 6025E, and DIO-32F and 653*X*s devices in the *DAQ Hardware Overview Guide* list the onboard port numbers used for each device type if the SCXI-1162/HV or SCXI-1163/R is configured for parallel mode. The MIO and AI devices, Lab-PC+, and SCXI-1200 cannot use the SCXI-1162/HV or the SCXI-1163/R in parallel mode.

4

NI-DAQ Double Buffering

This chapter describes using double-buffered data acquisitions with NI-DAQ. This chapter applies to counter operations. However, you can read samples of any size.

Overview

Conventional data acquisition software techniques, such as single-buffered data acquisition, work well for most applications. However, more sophisticated applications involving larger amounts of data 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 memory buffer stores the data, 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 outputting data, 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 operations is that the amount of data that can be input or output is limited to the amount of free memory available in the computer.

In double-buffered input 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 *indefinitely* 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.

Unlike single-buffered operations, double-buffered operations reuse the same buffer and are therefore able to input or output an infinite number of 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, storing, and processing. The next two sections explain how to access the data 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 4-1 illustrates a series of sequential data transfers.

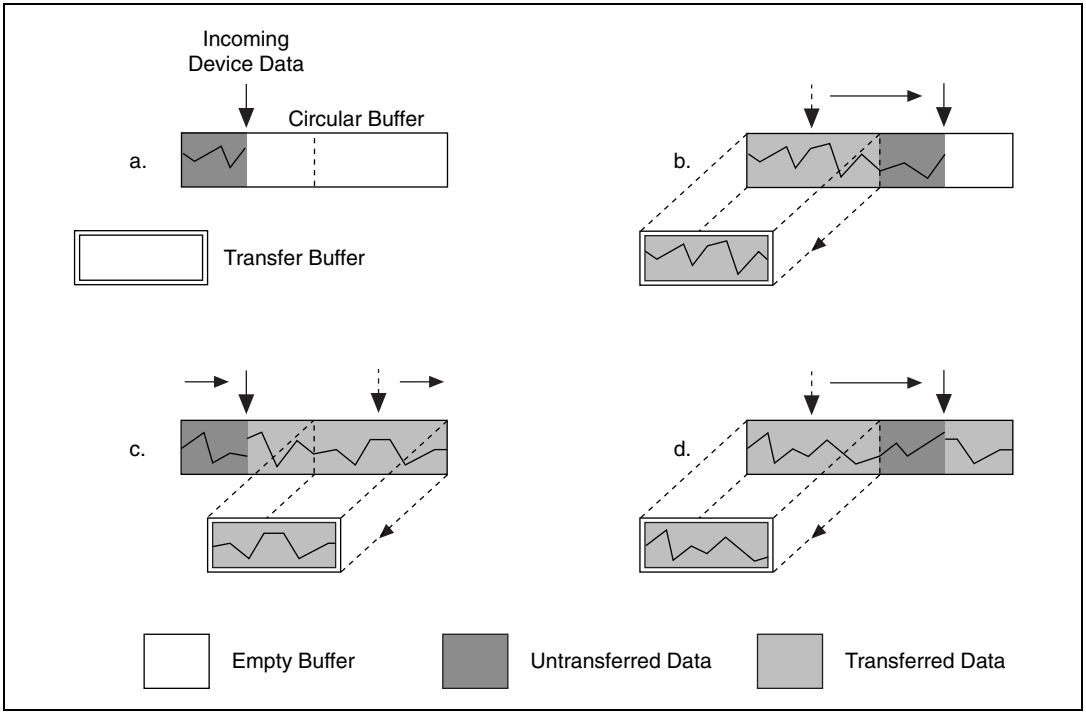


Figure 4-1. Double-Buffered Input with Sequential Data Transfers

The double-buffered input operation begins when the DAQ device starts writing data into the first half of the circular buffer (Figure 4-1a). 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 4-1b). You can then store the data in the transfer block to disk or process it according to your application needs. After the input device has filled the second half of the circular buffer, the device returns to the first half of the buffer and overwrites the old data. NI-DAQ can now copy the second half of the circular buffer to the transfer buffer (Figure 4-1c). 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. Notice that Figure 4-1d is equivalent to the step in Figure 4-1b and is the start of a two-step cycle.

Potential Setbacks

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 4-2.

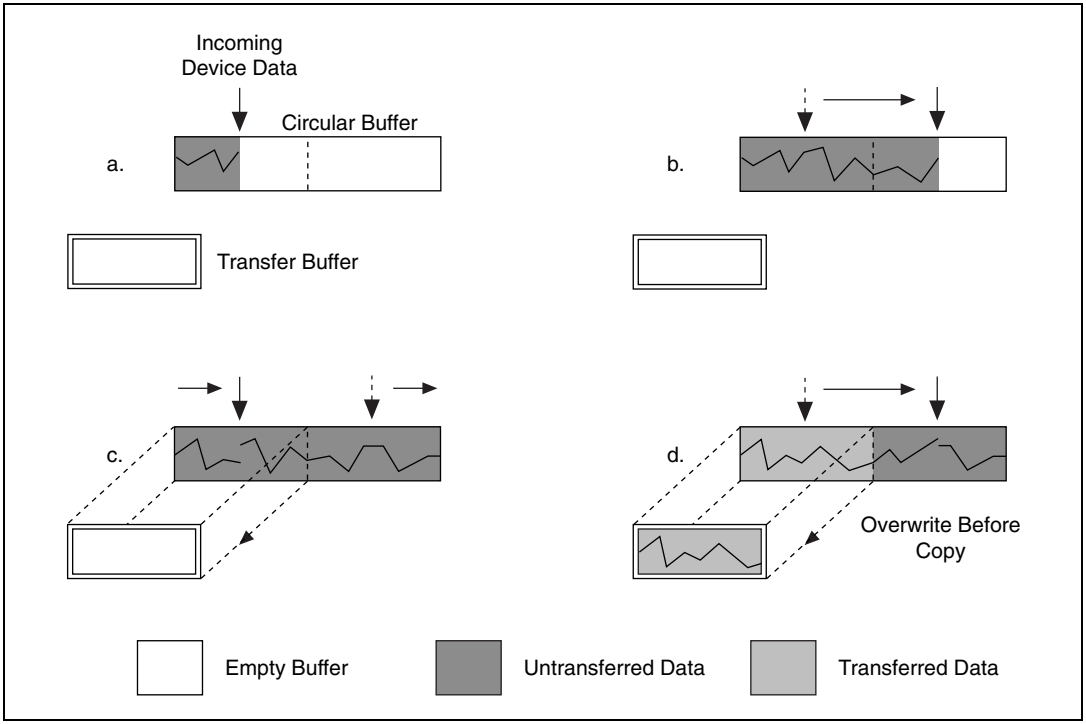


Figure 4-2. Double-Buffered Input with an Overwrite Before Copy

Notice that in Figure 4-2b, 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 overwriting the data in the first half of the circular buffer before NI-DAQ has copied it to the transfer buffer (Figure 4-2c). To guarantee uncorrupted data, NI-DAQ must 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 4-2d).

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 4-1.

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 4-3.

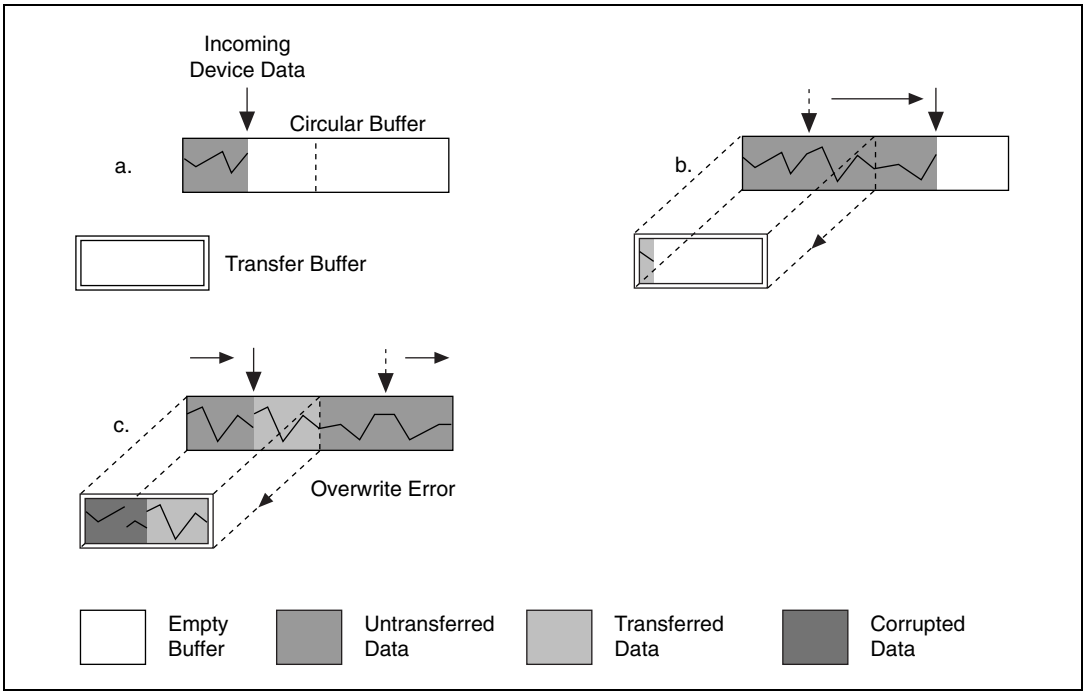


Figure 4-3. Double-Buffered Input with an Overwrite

In Figure 4-3b, 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 4-3c). 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 re-occur.

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 4-4 illustrates a series of sequential data transfers.

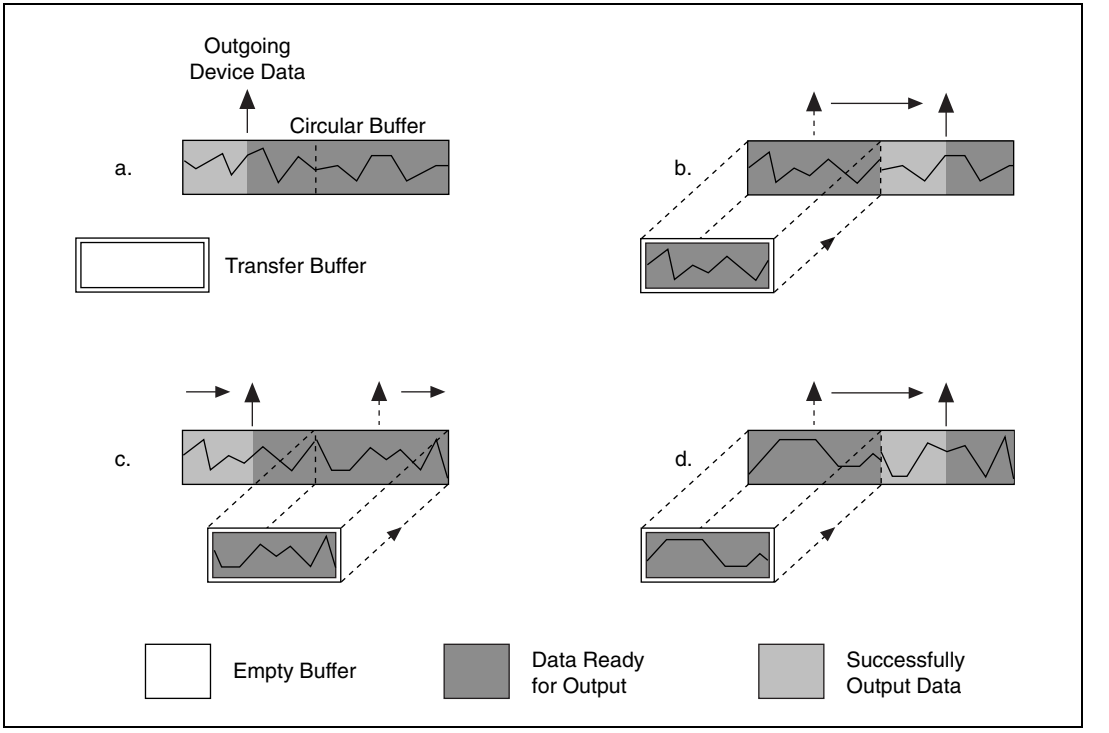


Figure 4-4. Double-Buffered Output with Sequential Data Transfers

The double-buffered output operation begins when the output device begins outputting data from the first half of the circular buffer (Figure 4-4a). 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 4-4b). Your application can then update the data in the transfer buffer. After the output device is 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 can now copy the transfer buffer to the second half of the circular buffer (Figure 4-4c). 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. Notice that Figure 4-4d is equivalent to the step in Figure 4-4b and is the start of a two-step cycle.

Potential Setbacks

Like double-buffered input, double-buffered output has 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 4-5.

Notice in Figure 4-5b, 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 4-5c). 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 4-5d).

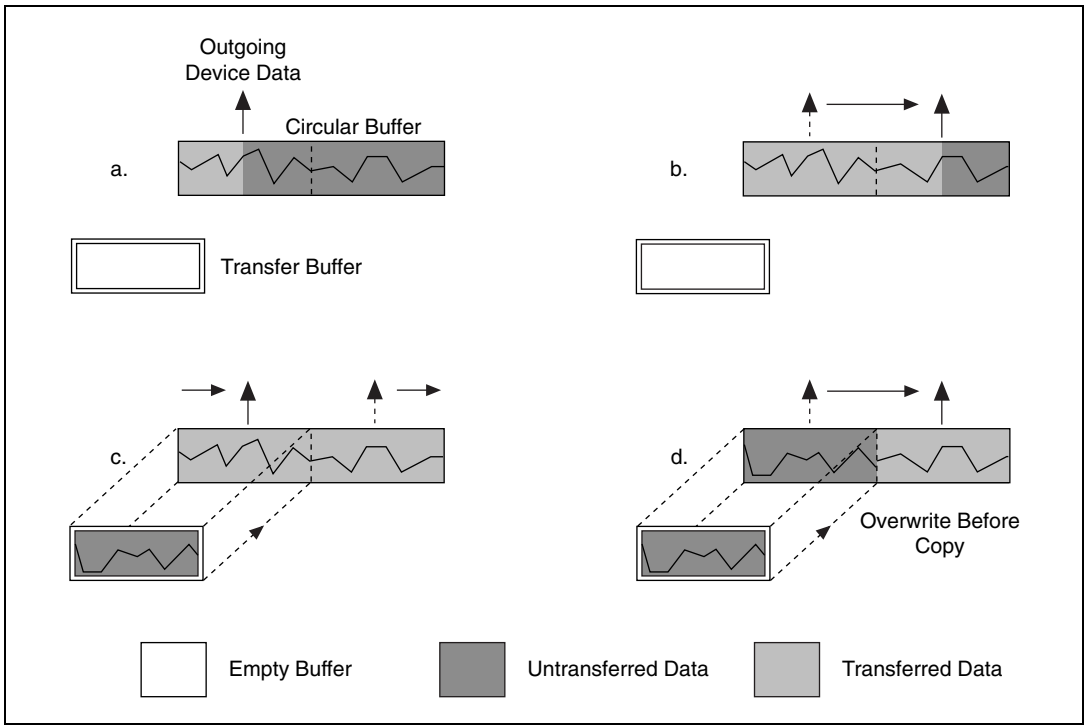


Figure 4-5. Double-Buffered Output with an Overwrite Before Copy

For this situation, NI-DAQ returns an overwrite before a 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 4-4.

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 4-6.

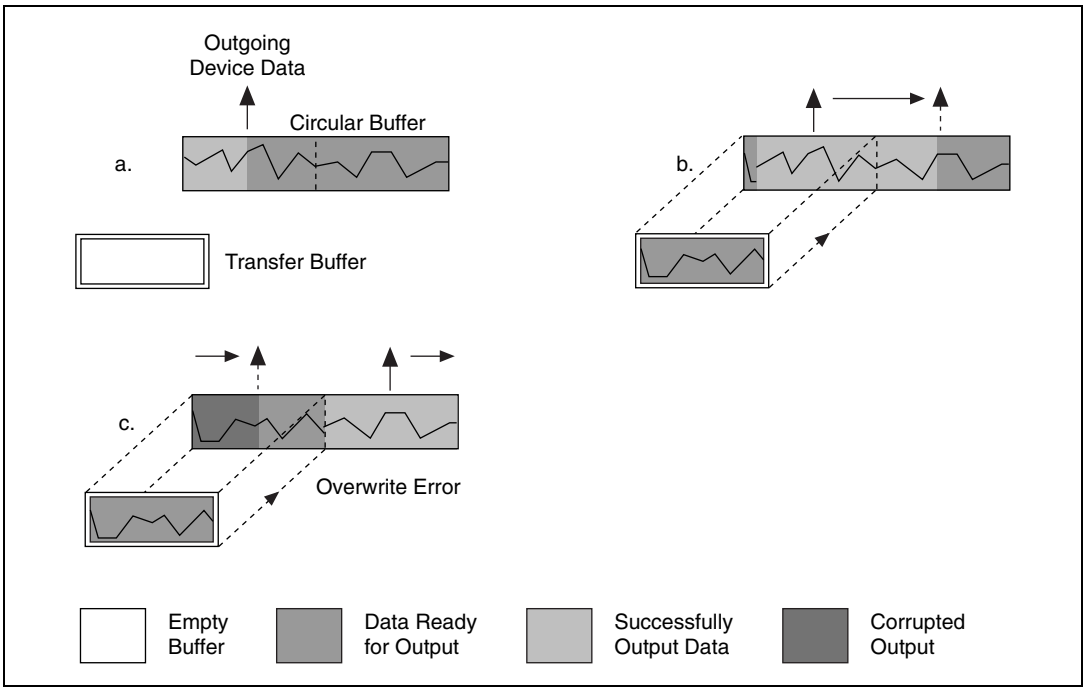


Figure 4-6. Double-Buffered Output with an Overwrite

In Figure 4-6b, 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 4-6c). Consequently, device data output might be corrupted; 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), digital input/output (DIG), and general-purpose counter (GPCTR) operations. This section explains what each of the functions do and the order in which you should call them.

Double Buffer Configuration Functions

The Double Buffer Configuration 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`
- `GPCTR_Change_Parameter`

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.

For counter operations, before calling `GPCTR_Control` to start your counter, call `GPCTR_Change_Parameter` to enable or disable continuous buffering (with `ND_Double`). Counters transfer their data continuously, not in half buffers.

Double Buffer Transfer Functions

After a double-buffered operation begins, the Double Buffer 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 Double Buffer Transfer functions also check for possible errors during the transfer.

For input operations, `DB_Transfer` copies data from alternating halves of the circular input buffer to 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 a 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 for DAQ, WFM, DIG, and GPCTR 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 for DAQ, WFM, DIG, and GPCTR 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. For counter operations, the transfer function takes timeout as one of the parameters, so you do need to call Timeout_Config prior to calling the transfer function.

The transfer functions are as follows:

- DAQ_DB_Transfer
- WFM_DB_Transfer
- DIG_DB_Transfer
- GPCTR_Read_Buffer

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.

For counter operations call GPCTR_Read_Buffer after starting the operation to transfer a specified portion of the double buffer.

Double Buffer HalfReady Functions

With the Double Buffer HalfReady functions, applications can avoid the delay possible when calling the double buffer 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 Double Buffer HalfReady functions check if a double buffer transfer can be completed immediately. If the call to Double Buffer 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
- GPCTR_Read_Buffer

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.

For counter operations, call GPCTR_Read_Buffer after calling GPCTR_Control with a **timeOut** of 0, to check the transfer status of the operation. Counters actually transfer their data in a continuous manner, not a double-buffered manner.

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.

5

Transducer Conversion Functions

This chapter describes the NI-DAQ Transducer Conversion functions. NI-DAQ includes source code for these functions.

The Transducer Conversion functions convert analog input voltages read from thermocouples, RTDs, thermistors, and strain gauges into temperature or strain units:

`RTD_Convert`	Supplied single-voltage and voltage-buffer routines convert voltages read from an RTD into resistance and then into temperature in units for Celsius, Fahrenheit, Kelvin, or Rankine.
`Strain_Convert`	Supplied single-voltage and voltage-buffer routines convert voltages read from a strain gauge into measured strain using the appropriate formula for the strain gauge bridge configuration used.
`Thermistor_Convert`	Supplied single-voltage and voltage-buffer routines convert voltages read from thermistors into temperature.
`Thermocouple_Convert`	Supplied single-voltage and voltage-buffer routines 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 in order to call the routines in your application.

The conversion routines are 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 your application's specific accuracy requirements. Comments in the conversion source code simplify the process of making only necessary changes.

A header file for each language (`convert.h` for C/C++, `convert.bas` for Visual Basic) 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.

Function Descriptions

RTD_Convert and RTD_Buf_Convert

These functions convert a voltage or voltage buffer that NI-DAQ reads from an RTD into temperature.

Parameter Discussion

The **convType** integer indicates whether to use the given conversion formula, or to use a user-defined formula that you have put into the routine.

0: The given conversion formula.

–1: Use a user-defined formula that has been added to the routine.

Iex is the excitation current in amps that was used with the RTD. If a 0 is passed in **Iex,** a default excitation current of 150×10^{-6}A (150 mA) is assumed.

Ro is the RTD resistance in ohms at 0 °C.

A and **B** are the coefficients of the Callendar Van-Düsen equation that fit your RTD.

The **TempScale** integer indicates 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-Düsen equation for RTDs:

$$R_t = R_0[1 + At + Bt^2 + C(t–100)t^3]$$

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 $\mathbf{R_0}$ and the **A** and **B** coefficients for the Callendar Van-Düsen 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):

  $\mathbf{A} = 3.90802 \times 10^{-3}$

  $\mathbf{B} = -5.80195 \times 10^{-7}$

  ($\alpha = 3.85 \times 10^{-3}$; $\partial = 1.492$)
- American Curve:

  $\mathbf{A} = 3.9784 \times 10^{-3}$

  $\mathbf{B} = -5.8408 \times 10^{-7}$

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

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

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

$$\mathbf{B} = -\alpha\partial/(10,000,100^2)$$

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

These functions convert a voltage or voltage buffer that NI-DAQ read from a strain gauge to units of strain.

Parameter Discussion

The **bridgeConfig** integer indicates in what type of bridge configuration the strain gauge is mounted. Figure 5-1 shows all the different bridge configurations and the corresponding values that you should pass in **bridgeConfig**.

Vex is the excitation voltage (in volts) 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 excitation voltages of 10 V and 3.333 V. The SCXI-1122 module provides 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 in ohms.

RL is the lead resistance in ohms. 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 in volts after it is mounted in its bridge configuration. 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.

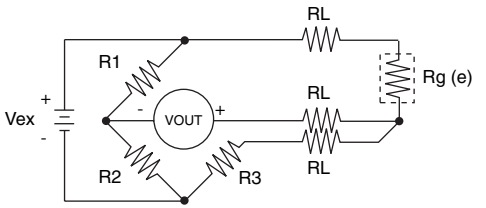
The `Strain_Buf_Convert` routine has three remaining parameters—**numPts** is the number of voltage points to convert, **strainVoltBuf** is the array that contains the voltages that NI-DAQ read from the strain gauge, and **strainValBuf** is the return array that contains the strain values.

Using This Function

The conversion formula used is based solely on the bridge configuration. Figure 5-1 shows the seven bridge configurations supported and the corresponding formulas. For all bridge configurations, NI-DAQ uses the following formula to obtain V_r:

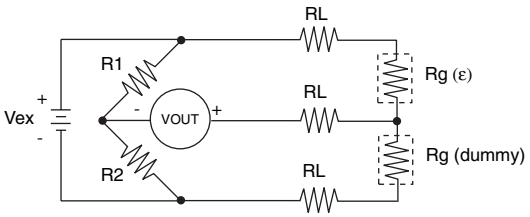
$$V_r = (\mathbf{strainVolts} - \mathbf{V_{init}}) / \mathbf{Vex}$$

In the circuit diagrams shown in Figure 5-1, 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, $\mathbf{R_1}$ and $\mathbf{R_2}$ are dummy resistors that are not directly incorporated into the conversion formula. The SCXI-1121 and SCXI-1122 modules provide $\mathbf{R_1}$ and $\mathbf{R_2}$ 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.



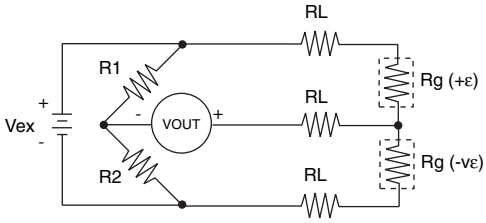
bridgeConfig = 1 (QTR_BRIDGE_I)

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



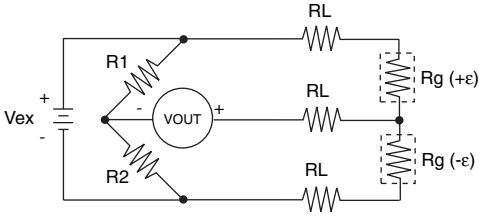
bridgeConfig = 2 (QTR_BRIDGE_II)

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



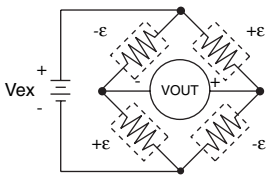
bridgeConfig = 3 (HALF_BRIDGE_I)

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



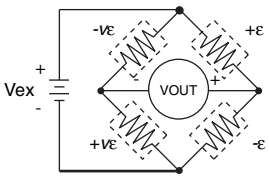
bridgeConfig = 4 (HALF_BRIDGE_II)

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



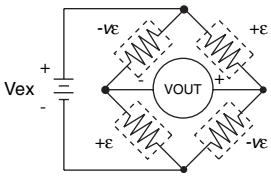
bridgeConfig = 5 (FULL_BRIDGE_I)

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



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 5-1. Strain Gauge Bridge Configuration

Thermistor_Convert and Thermistor_Buf_Convert

These functions convert a voltage or voltage buffer read from a thermistor into temperature. Some SCXI terminal blocks have onboard thermistors that you can use to do cold-junction compensation.

Parameter Discussion

V_{ref} is the voltage reference you apply across the thermistor circuit (see Figure 5-2) in volts. The thermistor on the SCXI terminal blocks has a **V_{ref}** of 2.5 V.

R_1 is the value of the resistor in series with your thermistor (see Figure 5-2) in ohms. The thermistor on the SCXI terminal blocks has an **R_1** value of 5,000 Ω.

The **TempScale** integer indicates 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 $\mathbf{R_t}$, the thermistor resistance (see Figure 5-2).

$$\mathbf{Volts} = \mathbf{V_{ref}} \left(\mathbf{R_t} / (\mathbf{R_1} + \mathbf{R_t}) \right)$$

Solving the previous equation for $\mathbf{R_t}$, we have:

$$\mathbf{R_t} = \mathbf{R_1} \left(\mathbf{Volts} / (\mathbf{V_{ref}} - \mathbf{Volts}) \right)$$

After this function calculates $\mathbf{R_t}$, the function uses the following equation to convert $\mathbf{R_t}$, the thermistor resistance, to temperature in Kelvin. The function then converts the temperature to the temperature scale you want, if necessary.

$$T = 1 / (\mathbf{a} + \mathbf{b}(\ln \mathbf{R_t}) + \mathbf{c}(\ln \mathbf{R_t})\text{^}3)$$

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.

$$\boldsymbol{a} = 1.295361\text{E}{-3}$$

The following equation expresses the relationship between **Volts** and $\mathbf{R_t}$, the thermistor resistance (see Figure 5-2).

$$\mathbf{Volts} = \mathbf{V_{ref}} \left(\mathbf{R_t} / (\mathbf{R_1} + \mathbf{R_t}) \right)$$

Solving the previous equation for $\mathbf{R_t}$, you have:

$$\mathbf{R_t} = \mathbf{R_1} \left(\mathbf{Volts} / (\mathbf{V_{ref}} - \mathbf{Volts}) \right)$$

When you calculate $\mathbf{R_t}$, you use the following equation to convert $\mathbf{R_t}$, the thermistor resistance, to temperature in Kelvin. Then convert the temperature to the temperature scale you want, if necessary.

$$T = 1 / (\mathbf{a} + \mathbf{b}(\ln \mathbf{R_t}) + \mathbf{c}(\ln \mathbf{R_t})\text{^}3)$$

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

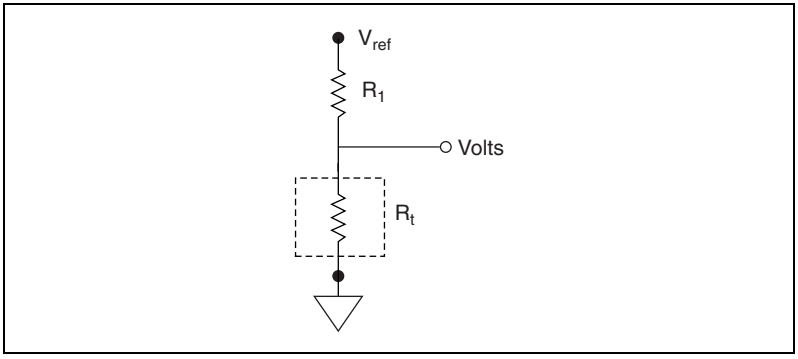


Figure 5-2. Circuit Diagram of a Thermistor in a Voltage Divider

Thermocouple_Convert and Thermocouple_Buf_Convert

These functions convert a voltage or voltage buffer that NI-DAQ read from a thermocouple into temperature.

Parameter Discussion

The **TCType** integer indicates 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, most likely this 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.

The **TempScale** integer indicates 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 5-1. All the reference equations and inverse equations used in these routines are from *NIST Monograph 175*.

Table 5-1 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 5-1. 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

A

Technical Support Resources

Web Support

National Instruments Web support is your first stop for help in solving installation, configuration, and application problems and questions. Online problem-solving and diagnostic resources include frequently asked questions, knowledge bases, product-specific troubleshooting wizards, manuals, drivers, software updates, and more. Web support is available through the Technical Support section of `ni.com`

NI Developer Zone

The NI Developer Zone at `zone.ni.com` is the essential resource for building measurement and automation systems. At the NI Developer Zone, you can easily access the latest example programs, system configurators, tutorials, technical news, as well as a community of developers ready to share their own techniques.

Customer Education

National Instruments provides a number of alternatives to satisfy your training needs, from self-paced tutorials, videos, and interactive CDs to instructor-led hands-on courses at locations around the world. Visit the Customer Education section of `ni.com` for online course schedules, syllabi, training centers, and class registration.

System Integration

If you have time constraints, limited in-house technical resources, or other dilemmas, you may prefer to employ consulting or system integration services. You can rely on the expertise available through our worldwide network of Alliance Program members. To find out more about our Alliance system integration solutions, visit the System Integration section of `ni.com`

Worldwide Support

National Instruments has offices located around the world to help address your support needs. You can access our branch office Web sites from the Worldwide Offices section of `ni.com`. Branch office web sites provide up-to-date contact information, support phone numbers, e-mail addresses, and current events.

If you have searched the technical support resources on our Web site and still cannot find the answers you need, contact your local office or National Instruments corporate. Phone numbers for our worldwide offices are listed at the front of this manual.

Glossary

Prefix	Meaning	Value
μ-	micro-	10^{-6}
m-	milli-	10^{-3}
k-	kilo-	10^{3}
M-	mega-	10^{6}

Symbols

ß	coefficient
∂	coefficient
°	degree
–	minus
Ω	ohm
%	percent
+	plus
±	plus or minus
ε	strain
α	temperature coefficient at T = 0 °C

A

A/D	analog-to-digital
AC	alternating current

ACK	acknowledge
ActiveX	A programming system and user interface that lets you work with interactive objects. Formerly called OLE.
ActiveX control	a standard software tool that adds additional functionality to any compatible ActiveX container
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.
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 (NI 54*XX* devices only)
asynchronous	(1) hardware—a property of an event that occurs at an arbitrary time, without synchronization to a reference clock (2) software—an action or event that occurs at an unpredictable time with respect to the execution of a program

B

background acquisition	Data is acquired by a DAQ system while another program or processing routine is running without apparent interruption
bandwidth	the range of frequencies present in a signal, or the range of frequencies to which a measuring device can respond

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 PCI bus, AT bus, 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 C/C++, Visual Basic (version 5.0), or Borland Delphi, 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	(1) data acquisition—collecting and measuring electrical signals from sensors, transducers, and test probes or fixtures and inputting them to a computer for processing (2) data acquisition—collecting and measuring the same kinds of electrical signals with A/D and/or DIO devices plugged into a computer, and possibly generating control signals with D/A and/or DIO devices in the same computer
DC	direct current
DDS	Direct Digital Synthesis
device	A plug-in DAQ board, card, or pad that can contain multiple channels and conversion devices. Plug-in boards, PCMCIA cards, and devices such as the DAQPad-1200, which connects to your computer parallel port, are all examples of DAQ devices. SCXI modules are distinct from devices, with the exception of the SCXI-1200, which is a hybrid.
differential input	an analog input consisting of two terminals, both of which are isolated from computer ground, whose difference is measured
digital port	*See* port
DIN	Deutsche Industrie Norme (German Industrial Standard)
DIO	digital I/O
dithering	the addition of Gaussian noise to the analog input signal

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
DSA	dynamic signal acquisition
DSP	digital signal processing
DSUB	D-subminiature connector

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
event-driven message	a message sent by an event-driven program, a program with a loop that waits for events to occur
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
GPS	Global Positioning System. A satellite-based system created and maintained by the U.S. Department of Defense that allows its users to determine their position, velocity, and time.
GPS receiver	An instrument that receives signals from GPS satellites
GUI	graphical user interface

H

Hz	hertz

I

I/O	input/output
ID	identification
IDE	Integrated Development Environment

IEEE	Institute of Electrical and Electronics Engineers
import library	a file that contains information the linker needs to resolve external references to exported dynamic link library (DLL) functions, so the system can locate the specified DLL and exported DLL functions at run time
interrupt	a computer signal indicating that the CPU should suspend its current task to service a designated activity
interrupt latency	the delay between the time hardware asserts an interrupt and when the interrupt service routine is activated
IRIG	Inter Range Instrumentation Group. A pulse-modulated signal normally produced by a GPS receiver.
IRQ	interrupt request
ISA	Industry Standard Architecture
iterations	repetitions of the buffer

J

Julian	a measurement of time based on the Julian calendar, a commonly used calendar that divides each year into 12 months with 365 days

K

kS	1,000 samples
Kword	1,024 words of memory

L

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

master/slave	Type of network connection in which a request is transmitted to one or more destination nodes, and those nodes send a response back to the requesting node. In industrial applications, the responding (slave) device is usually a sensor or actuator, and the requesting (master) device is usually a controller.
MB	megabytes of memory
MIO	multifunction I/O
MS	million samples
MSB	most significant bit
multirate scanning	scanning different channels at different rates
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
NO	normally open

O

OCX	OLE Control eXtension. Another name for OLE or ActiveX controls, reflected by the .OCX file extension of ActiveX control files.
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

PCI	Peripheral Component Interconnect
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
PPS	Pulse per second. A signal normally produced by a GPS receiver.
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
PXI	PCI eXtensions for Instrumentation. PXI is an open specification that builds off the CompactPCI specification by adding instrumentation-specific features.

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% of full scale.
ROM	read-only memory
RTC	Real time clock. A clock capable of recording the exact time of events that counts time in days, hours, minutes, seconds, and fractions of seconds.

RTD	Resistive 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 devices directly, by means of connectors on top of the boards, for precise synchronization of functions.

S

S/s	samples per second—used to express the rate at which a DAQ device samples an analog signal
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 device that has an extremely stable onboard reference and calibrates its own A/D and D/A circuits without manual adjustments by the user
Single-Ended (SE) Inputs	an analog input that is measured with respect to a common ground
slave	*See* master/slave
software trigger	a programmed event that triggers an event such as data acquisition
stage	(NI 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
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)
USB	universal serial bus

V

V	volt
VDC	volts direct current
VPICD	Virtual Programmable Interrupt Controller Device
VXI	VMEbus eXtensions for Instrumentation

Index

Numbers

1200 series devices. *See* Lab and 1200 series devices.
4451and 4551 devices
  counter usage, 3-50
  FIFO lag effect, 3-51 to 3-52
6025E groups of ports, 3-55
6115 and 6120 devices. *See* PCI-6115 and PCI-6120 devices.
622*X* devices
  counter usage, 3-50
  FIFO lag effect, 3-51 to 3-52
652*X* devices
  digital change detection applications, 3-60
  digital change notification applications, 3-60
653*X* devices
  digital change detection applications, 3-61 to 3-62
  groups of ports, 3-55 to 3-56
  pattern generation, 3-74 to 3-75
  RTSI connections, 3-105 to 3-106
660*X* devices, RTSI connections, 3-103
671*X* devices
  counter usage, 3-50
  FIFO lag effect, 3-51 to 3-52
  RTSI connections, 3-103

A

ACK1 signal
  653*X* RTSI connections (table), 3-105
  DIO-32F RTSI connections (table), 3-104
ACK2 signal
  653*X* RTSI connections (table), 3-105
  DIO-32F RTSI connections (table), 3-104
ActiveX controls for Visual Basic, 3-6 to 3-7
AI device terminology (table), *xv-xvii*
AI_Change_Parameter function, 3-16
AI_Check function, 3-16
AI_Clear function, 3-16
AI_Configure function, 3-16
AI_Mux_Config function, 3-17
AI_Read function, 3-17
AI_Read_Scan function, 3-17
AI_Read_VScan function, 3-17
AI_Setup function, 3-17
AI_VRead function, 3-17
AI_VScale function, 3-17
Align_DMA_Buffer function, 3-2
Am9513-based devices
  gating modes, 3-79
  timebases, 3-78
AMUX-64T external multiplexer support, 3-25
Analog Alarm Event control, 3-11 to 3-13
  properties (table), 3-11
  setting program flow, 3-12 to 3-13
  setting properties, 3-12
analog input application tips. *See* data acquisition application tips.
analog input functions. *See* data acquisition functions; one-shot analog input functions.
analog output application tips, 3-36 to 3-38
  equivalent analog output calls (figure), 3-36
  SCXI applications, 3-126
  simple application, 3-37
  software update application, 3-37 to 3-38
analog output devices, reference voltages for (tables), 3-49 to 3-50
analog output functions, 3-35 to 3-36. *See also* waveform generation functions.
  AO_Change_Parameter, 3-35
  AO_Configure, 3-35
  AO_Update, 3-35

AO_VScale, 3-36
AO_VWrite, 3-36
AO_Write, 3-36
Analog Trigger Event control, 3-9 to 3-11
properties (table), 3-9 to 3-10
setting properties, 3-10 to 3-11
AO_Calibrate function, 3-3
AO_Change_Parameter function, 3-35
AO_Configure function, 3-35
AO_Update function, 3-35
AO_VScale function, 3-36
AO_VWrite function, 3-36
AO_Write function, 3-36
applications for Windows. *See* building Windows applications.
AT-AO-6/10
counter usage in waveform generation, 3-51
FIFO lag effect, 3-51 to 3-52
reference voltages for waveform generation (tables), 3-50
RTSI bus connections, 3-104
AT-DIO-32F. *See* DIO-32F.
AT-MIO-16DE-10, groups of ports, 3-55

B

Borland C++ Windows applications, 2-7 to 2-8
Borland Delphi, 2-8
buffer allocation in Windows applications
Microsoft Visual Basic, 2-5 to 2-6
Microsoft Visual C++, 2-3
building Windows applications, 2-1 to 2-10
Borland C++, 2-7 to 2-8
Microsoft Visual Basic, 2-4 to 2-6
Microsoft Visual C++, 2-2 to 2-3
NI-DAQ examples, 2-9 to 2-10
NI-DAQ libraries, 2-1

C

Calibrate_1200 function, 3-3 to 3-4
Calibrate_DSA function, 3-4
Calibrate_E_Series function, 3-4
Calibrate_TIO function, 3-4
calibration functions. *See* software-calibration and device-specific functions.
clocks or time counters for NI-TIO devices, 3-95 to 3-102
accuracy, 3-98
clock resolution, 3-95
example clock in measurement system, 3-98 to 3-99
sample use cases, 3-99 to 3-102
synchronization, 3-96 to 3-97
IRIG (Inter Range Instrumentation Group) protocols, 3-97 to 3-98
pulse per second, 3-96 to 3-97
real-time clock synchronizing to GPS receiver (figure), 3-95
sample code for setting up clock using PPS, 3-96 to 3-97
concatenated event counting, 3-88 to 3-89
Config_Alarm_Deadband function, 3-5
Config_ATrig_Event_Message function, 3-5
Config_DAQ_Event_Message function, 3-5
configuration
NI-DAQ hardware support (tables), 1-4 to 1-6
using Measurement & Automation Explorer, 1-7
configuration functions. *See also* initialization and general-configuration functions.
AI_Change_Parameter, 3-16
AI_Configure, 3-16
AI_Mux_Config, 3-17
AI_Setup, 3-17
AO_Configure, 3-35
Config_Alarm_Deadband, 3-5
Config_ATrig_Event_Message, 3-5

Config_DAQ_Event_Message, 3-5
Configure_HW_Analog_Trigger, 3-4
CTR_Config, 3-77
CTR_FOUT_Config, 3-77
DAQ_Config, 3-21
DAQ_DB_Config, 3-23, 4-10
DAQ_StopTrigger_Config, 3-22
DIG_Block_PG_Config, 3-58
DIG_Change_Message_Config, 3-60
DIG_DB_Config, 3-59, 4-10
DIG_Filter_Config, 3-60
DIG_Grp_Config, 3-58
DIG_Line_Config, 3-57
DIG_Prt_Config, 3-58
DIG_SCAN_Setup, 3-59
DIG_Trigger_Config, 3-59
GPCTR_Change_Parameter, 3-93, 4-10
MIO_Config, 3-4
SCXI_Configure_Connection, 3-108
SCXI_Configure_Filter, 3-108
SCXI_Load_Config, 3-109
SCXI_MuxCtr_Setup, 3-110
SCXI_SCAN_Setup, 3-110
SCXI_Set_Config, 3-110
SCXI_Set_Excitation, 3-110
SCXI_Set_Gain, 3-111
SCXI_Set_Input_Mode, 3-111
SCXI_Set_State, 3-111
SCXI_Set_Threshold, 3-111
SCXI_Single_Chan_Setup, 3-111
SCXI_Track_Hold_Setup, 3-111
Timeout_Config, 3-3
WFM_DB_Config, 3-40, 4-10
WFM_Group_Setup, 3-40
Configure_HW_Analog_Trigger function, 3-4
conventions used in manual, *xi-xvii*
counter/timer application tips, 3-82 to 3-91. *See also* counter/timer functions; counter/timer operation.
concatenated event counting, 3-88 to 3-89
event counting, 3-86 to 3-89
event counting flow chart (figure), 3-82
event-counting functions, 3-82
frequency measurement, 3-87 to 3-88
general-purpose counter/timer functions, 3-94 to 3-95
interval counter/timer functions, 3-92
period and continuous pulse-width measurement, 3-89 to 3-91
pulse generation flow chart (figure), 3-83
pulse width measurement, 3-86 to 3-87
pulse-generation functions, 3-83 to 3-84
simultaneous counter operations (figure), 3-85
time-lapse measurement, 3-87
timer event counting (figure), 3-86
utility functions, 3-84
counter/timer functions, 3-76 to 3-78. *See also* counter/timer application tips; counter/timer operation.
CTR_Config, 3-77
CTR_EvCount, 3-77
CTR_EvRead, 3-77
CTR_FOUT_Config, 3-77
CTR_Period, 3-77
CTR_Pulse, 3-77
CTR_Rate, 3-77
CTR_Reset, 3-77
CTR_Restart, 3-77
CTR_Simul_Op, 3-77
CTR_Square, 3-77
CTR_State, 3-78
CTR_Stop, 3-78
device support (table), 3-76
general-purpose counter/timer functions
application tips, 3-94 to 3-95
GPCTR_Change_Parameter, 3-93, 4-10
GPCTR_Config_Buffer, 3-93
GPCTR_Control, 3-93
GPCTR_Read_Buffer, 3-93, 4-11, 4-12

GPCTR_Set_Application, 3-93
GPCTR_Watch, 3-93
interval counter/timer functions
application tips, 3-92
ICTR_Read, 3-91
ICTR_Reset, 3-91
ICTR_Setup, 3-91
interval counter/timer block diagram, 3-92
counter/timer operation. *See also* interval counter/timer functions; interval counter/timer operation.
Am9513-based devices
gating modes, 3-79
timebases, 3-78
clocks or time counters, 3-95 to 3-102
clock accuracy, 3-98
clock resolution, 3-95
clock synchronization, 3-96 to 3-97
example clock in measurement system, 3-98 to 3-99
IRIG (Inter Range Instrumentation Group) protocols, 3-97 to 3-98
pulse per second, 3-96 to 3-97
sample code for setting up clock using PPS, 3-96 to 3-97
sample use cases, 3-99 to 3-102
CTR function operation, 3-78 to 3-81
counter block diagram, 3-78
counter timing and output types (figure), 3-81
programmable frequency output operation, 3-81
data acquisition function application tips, 3-24 to 3-25
interval counter/timer operation, 3-92
counter usage, in waveform generation, 3-50 to 3-51
CTR_Config function, 3-77
CTR_EvCount function, 3-77
CTR_EvRead function, 3-77
CTR_FOUT_Config function, 3-77
CTR_Period function, 3-77
CTR_Pulse function, 3-77
CTR_Rate function, 3-77
CTR_Reset function, 3-77
CTR_Restart function, 3-77
CTR_Simul_Op function, 3-77
CTR_Square function, 3-77
CTR_State function, 3-78
CTR_Stop function, 3-78
customer education, A-1

D

DAQ system, setting up, 1-2 to 1-3
DAQCard-500/700 devices
data acquisition application tips, 3-25
interval counter/timer operation, 3-91 to 3-92
DAQ_Check function, 3-21
DAQ_Clear function, 3-21
DAQ_Config function, 3-21
DAQ_DB_Config function, 3-23, 4-10
DAQ_DB_HalfReady function, 3-24, 4-12
DAQ_DB_Transfer function, 3-24, 4-11
DAQ_Monitor function, 3-21
DAQ_Op function, 3-20
DAQ_Rate function, 3-22
DAQ_Set_clock function, 3-22
DAQ_Start function, 3-22
DAQ_StopTrigger_Config function, 3-22
DAQ_to_Disk function, 3-20
DAQ_VScale function, 3-22
data acquisition application tips, 3-24 to 3-32
basic building blocks, 3-25 to 3-26
building block 1: configuration functions, 3-26
building block 2: start functions, 3-27 to 3-29
building block 3: check functions, 3-30 to 3-31

building block 4: cleaning up, 3-31
DAQCard-500/700, 516 devices, and LPM device counter/timer signals, 3-25
double-buffered data acquisition, 3-31 to 3-32
external multiplexer support (AMUX-64T), 3-25
Lab and 1200 device counter/timer signals, 3-24
data acquisition functions
high-level data acquisition functions, 3-20 to 3-21
DAQ_Op, 3-20
DAQ_to_Disk, 3-20
Lab_ISCAN_Op, 3-20
Lab_ISCAN_to_Disk, 3-20
SCAN_Op, 3-21
SCAN_to_Disk, 3-21
low-level data acquisition functions, 3-21 to 3-23
DAQ_Check, 3-21
DAQ_Clear, 3-21
DAQ_Config, 3-21
DAQ_Monitor, 3-21
DAQ_Rate, 3-22
DAQ_Set_clock, 3-22
DAQ_Start, 3-22
DAQ_StopTrigger_Config, 3-22
DAQ_VScale, 3-22
Lab_ISCAN_Check, 3-22
Lab_ISCAN_Start, 3-22
SCAN_Demux, 3-22
SCAN_Sequence_Demux, 3-22 to 3-23, 3-33
SCAN_Sequence_Retrieve, 3-23, 3-33
SCAN_Sequence_Setup, 3-23, 3-33
SCAN_Setup, 3-23
SCAN_Start, 3-23
low-level double-buffered data acquisition functions, 3-23 to 3-24
DAQ_DB_Config, 3-23, 4-10
DAQ_DB_HalfReady, 3-24, 4-12
DAQ_DB_Transfer, 3-24, 4-11
multirate scanning, 3-32 to 3-34
data acquisition rates, SCXI modules, 3-124 to 3-126
device configuration
NI-DAQ hardware support (tables), 1-4 to 1-6
using Measurement & Automation Explorer, 1-7
devices. *See also* specific device, e.g., AT-AO-6/10.
MIO and AI device terminology (table), *xv-xvii*
reference voltages for analog output devices (tables), 3-49 to 3-50
DIG_Block_Check function, 3-58
DIG_Block_Clear function, 3-58
DIG_Block_In function, 3-58
DIG_Block_Out function, 3-58
DIG_Block_PG_Config function, 3-58
DIG_Change_Message_Config function, 3-60
DIG_Change_Message_Control function, 3-60
DIG_DB_Config function, 3-59, 4-10
DIG_DB_HalfReady function, 3-59, 4-12
DIG_DB_Transfer function, 3-59, 4-11
DIG_Filter_Config function, 3-60
DIG_Grp_Config function, 3-58
DIG_Grp_Mode function, 3-58
DIG_Grp_Status function, 3-58
DIG_In_Grp function, 3-58
DIG_In_Line function, 3-57
DIG_In_Prt function, 3-57
digital I/O application tips, 3-62 to 3-76
digital change detection
with 652*X* devices, 3-60
with 653*X* devices, 3-61 to 3-62

digital change notification with 652*X* devices, 3-60
digital double-buffered group block I/O, 3-71 to 3-73
digital group block I/O, 3-69 to 3-70
digital group I/O, 3-67 to 3-68
digital line I/O, 3-65 to 3-66
digital port I/O applications, 3-63 to 3-65
double-buffered I/O, 3-75 to 3-76
handshaking versus no-handshaking digital I/O, 3-63
pattern generation I/O with DIO-32F, DIO 653*X,* PCI-6115, and PCI-6120 devices, 3-74 to 3-75
SCXI applications, 3-127
digital I/O functions, 3-52 to 3-60
byte mapping to digital I/O lines (table), 3-53 to 3-54
DIG_In_Line, 3-57
DIG_In_Prt, 3-57
digital change notification functions, 3-60
digital filtering function, 3-60
DIG_Line_Config, 3-57
DIG_Out_Line, 3-57
DIG_Out_Prt, 3-58
DIG_Prt_Config, 3-58
DIG_Prt_Status, 3-58
double-buffered digital I/O functions, 3-59
DIG_DB_Config, 3-59, 4-10
DIG_DB_HalfReady, 3-59, 4-12
DIG_DB_Transfer, 3-59, 4-11
group digital I/O functions, 3-58 to 3-59
DIG_Block_Check, 3-58
DIG_Block_Clear, 3-58
DIG_Block_In, 3-58
DIG_Block_Out, 3-58
DIG_Block_PG_Config, 3-58
DIG_Grp_Config, 3-58
DIG_Grp_Mode, 3-58
DIG_Grp_Status, 3-58
DIG_In_Grp, 3-58
DIG_Out_Grp, 3-59
DIG_SCAN_Setup, 3-59
DIG_Trigger_Config, 3-59
groups of ports
DIO-24, 6025E, AT-MIO-16DE-10, DIO-96, and Lab and 1200 series, 3-55
DIO-32F and 653*X,* 3-55 to 3-56
overview, 3-52 to 3-55
DIG_Line_Config function, 3-57
DIG_Out_Grp function, 3-59
DIG_Out_Line function, 3-57
DIG_Out_Prt function, 3-58
DIG_Prt_Config function, 3-58
DIG_Prt_Status function, 3-58
DIG_SCAN_Setup function, 3-59
DIG_Trigger_Config function, 3-59
DIO-24 groups of ports, 3-55
DIO-32F
groups of ports, 3-55 to 3-56
pattern generation, 3-74 to 3-75
RTSI bus connections, 3-104 to 3-105
DIO-96 groups of ports, 3-55
documentation
conventions used in manual, *xi-xvii*
how to use NI-DAQ manual set, *xi*
double-buffered data acquisition, 4-1 to 4-12
application tips, 3-31 to 3-32
input operations, 4-2 to 4-5
problem situations, 4-4 to 4-5
output operations, 4-6 to 4-9
problem situations, 4-7 to 4-9
overview, 4-1
single-buffered versus double-buffered data, 4-1 to 4-2
double-buffered functions
configuration functions, 4-10
DAQ_DB_Config, 3-23, 4-10
DIG_DB_Config, 3-59, 4-10

GPCTR_Change_Parameter, 3-93, 4-10
WFM_DB_Config, 4-10
digital I/O functions
applications
double-buffered I/O, 3-75 to 3-76
group block I/O applications, 3-69 to 3-70
DIG_DB_Config, 3-59, 4-10
DIG_DB_HalfReady, 3-59, 4-12
DIG_DB_Transfer, 3-59, 4-11
HalfReady functions, 4-11 to 4-12
DAQ_DB_HalfReady, 3-24, 4-12
DIG_DB_HalfReady, 3-59, 4-12
GPCTR_Read_Buffer, 4-12
WFM_DB_HalfReady, 4-12
low-level double-buffered data acquisition functions, 3-23 to 3-24
application tips, 3-31 to 3-32
DAQ_DB_Config, 3-23, 4-10
DAQ_DB_HalfReady, 3-24, 4-12
DAQ_DB_Transfer, 3-24, 4-11
transfer functions, 4-10 to 4-11
DAQ_DB_Transfer, 3-24, 4-11
DIG_DB_Transfer, 3-59, 4-11
GPCTR_Read_Buffer, 4-11
WFM_DB_Transfer, 4-11
double-buffered waveform generation applications, 3-47 to 3-48
DSA devices, RTSI connections, 3-103

E

E series devices
Calibrate_E_Series function, 3-4
counter usage, 3-50
FIFO lag effect, 3-51 to 3-52
RTSI bus connections, 3-103
event counting, 3-86 to 3-89
concatenated event counting, 3-88 to 3-89
CTR_EvCount function, 3-77
CTR_EvRead function, 3-77
flow chart, 3-82
frequency measurement, 3-87 to 3-88
overview, 3-86
pulse width measurement, 3-86 to 3-87
time-lapse measurement, 3-87
timer event counting (figure), 3-86
event message functions, 3-5. *See also* NI-DAQ events in Visual Basic for Windows.
application tips, 3-5 to 3-6
Config_Alarm_Deadband, 3-5
Config_ATrig_Event_Message, 3-5
Config_DAQ_Event_Message, 3-5
event procedures, 3-6
external device support (table)
parallel port, 1394, and other devices, 1-6
SCXI and USB devices, 1-6
external multiplexer support (AMUX-64T), 3-25
external triggering of waveform generation, 3-52
EXTUPD* signal (table), 3-104
EXTUPDATE* signal (table), 3-104

F

FIFO lag effect, 3-51 to 3-52
frequency measurement, 3-87 to 3-88
frequency output, counter/timer functions, 3-81
functions
analog input functions
one-shot analog input functions, 3-16 to 3-19
single-channel analog input application tips, 3-17 to 3-19
single-channel analog input functions, 3-16 to 3-19

analog output functions
application tips, 3-36 to 3-38
list of functions, 3-35 to 3-36
counter/timer functions
application tips, 3-82 to 3-91
counter/timer operation, 3-78 to 3-81
device support (table), 3-76
general-purpose counter/timer functions, 3-93
interval counter/timer functions, 3-91 to 3-92
list of functions, 3-77 to 3-78
programmable frequency output operation, 3-81
data acquisition functions
application tips, 3-24 to 3-32
double-buffered data acquisition application tips, 3-31 to 3-32
high-level data acquisition functions, 3-20 to 3-21
low-level data acquisition functions, 3-21 to 3-23
low-level double-buffered data acquisition functions, 3-23 to 3-24
digital I/O function group, 3-52 to 3-60
application tips, 3-62 to 3-76
byte mapping to digital I/O lines (table), 3-53 to 3-54
digital I/O functions, 3-57 to 3-58
digital line I/O applications, 3-65 to 3-66
DIO-24, 6025E, AT-MIO-16DE-10, DIO-96, and Lab and 1200 series groups, 3-55
DIO-32F and 653*X* groups, 3-55 to 3-56
double-buffered digital I/O functions, 3-59
group digital I/O functions, 3-58 to 3-59
overview, 3-52 to 3-55
event message functions
application tips, 3-5 to 3-6
list of functions, 3-5
NI-DAQ events in Visual Basic for Windows, 3-6 to 3-15
initialization and general configuration functions, 3-2 to 3-3
interval counter/timer functions
application tips, 3-92
interval counter/timer operation, 3-92
list of functions, 3-91
list of function groups, 3-1 to 3-2
RTSI bus trigger functions
653*X* RTSI connections, 3-105 to 3-106
application tips, 3-106 to 3-107
AT-AO-6/10 RTSI connections, 3-104
DIO-32F RTSI connections, 3-104 to 3-105
E series, DSA, 660*X,* and 671*X* RTSI connections, 3-103
list of functions, 3-102
SCXI functions
application tips, 3-112 to 3-124
list of functions, 3-107 to 3-111
software-calibration and device-specific functions, 3-3 to 3-4
transducer conversion functions
function descriptions, 5-2 to 5-11
list of functions, 5-1
overview, 5-1 to 5-2
waveform generation functions
application tips, 3-41 to 3-52
high-level waveform generation functions, 3-39
low-level waveform generation functions, 3-39 to 3-41

G

GATE2 signal (table), 3-104
gated pulse generation, 3-83 to 3-84
gating modes, Am9513-based devices, 3-79
General DAQ Event controls, 3-7 to 3-9
  examples, 3-13 to 3-15
  properties (table), 3-7 to 3-8
  setting properties, 3-8 to 3-9
general-purpose counter/timer functions
  application tips, 3-94 to 3-95
  GPCTR_Change_Parameter, 3-93, 4-10
  GPCTR_Config_Buffer, 3-93
  GPCTR_Control, 3-93
  GPCTR_Read_Buffer, 3-93
  GPCTR_Set_Application, 3-93
  GPCTR_Watch, 3-93
Get_DAQ_Device_Info function, 3-2
Get_NI_DAQ_Version function, 3-2
GPCTR_Change_Parameter function, 3-93, 4-10
GPCTR_Config_Buffer function, 3-93
GPCTR_Control function, 3-93
GPCTR_Read_Buffer function, 3-93, 4-11, 4-12
GPCTR_Set_Application function, 3-93
GPCTR_Watch function, 3-93
group digital I/O applications
  digital double-buffered group block I/O, 3-71 to 3-73
  digital group block I/O, 3-69 to 3-70
  digital group I/O, 3-67 to 3-68
group digital I/O functions, 3-58 to 3-59
  DIG_Block_Check, 3-58
  DIG_Block_Clear, 3-58
  DIG_Block_In, 3-58
  DIG_Block_Out, 3-58
  DIG_Block_PG_Config, 3-58
  DIG_Grp_Config, 3-58
  DIG_Grp_Mode, 3-58
  DIG_Grp_Status, 3-58
  DIG_In_Grp, 3-58
  DIG_Out_Grp, 3-59
  DIG_SCAN_Setup, 3-59
  DIG_Trigger_Config, 3-59
groups of ports
  DIO-24, 6025E, AT-MIO-16DE-10, DIO-96, and Lab and 1200 series groups, 3-55
  DIO-32F and 653*X* groups, 3-55 to 3-56
  PCI-6115 and PCI-6120 groups, 3-56 to 3-57

H

handshaking mode, 3-54 to 3-55
handshaking versus no-handshaking digital I/O, 3-63
hardware support (tables), 1-4 to 1-6
high-level data acquisition functions, 3-20 to 3-21
  application tips. *See* data acquisition application tips.
  DAQ_Op, 3-20
  DAQ_to_Disk, 3-20
  Lab_ISCAN_Op, 3-20
  Lab_ISCAN_to_Disk, 3-20
  SCAN_Op, 3-21
  SCAN_to_Disk, 3-21
high-level waveform generation functions, 3-39
  WFM_from_Disk, 3-39
  WFM_Op, 3-39

I

ICTR_Read function, 3-91
ICTR_Reset function, 3-91
ICTR_Setup function, 3-91
Init_DA_Brds function, 3-2

initialization and general-configuration functions
Align_DMA_Buffer, 3-2
Get_DAQ_Device_Info, 3-2
Get_NI_DAQ_Version, 3-2
Init_DA_Brds, 3-2
Set_DAQ_Device_Info, 3-3
Timeout_Config, 3-3
installation. *See also* device configuration.
flowchart for setting up your system, 1-3
setting up your DAQ system, 1-2 to 1-3
Inter Range Instrumentation Group (IRIG) time code protocols, 3-97 to 3-98
interval counter/timer functions, 3-91 to 3-92
application tips, 3-92
device support (table), 3-76
ICTR_Read, 3-91
ICTR_Reset, 3-91
ICTR_Setup, 3-91
interval counter/timer operation. *See also* counter/timer operation.
block diagram, 3-92
ICTR functions, 3-92
IRIG (Inter Range Instrumentation Group) time code protocols, 3-97 to 3-98

L

Lab and 1200 series devices
counter usage in waveform generation, 3-50 to 3-51
data acquisition application tips, 3-24
groups of ports, 3-55
interval counter/timer operation, 3-92
reference voltages for waveform generation (tables), 3-50
Lab_ISCAN_Check function, 3-22
Lab_ISCAN_Op function, 3-20
Lab_ISCAN_Start function, 3-22
Lab_ISCAN_to_Disk function, 3-20
languages supported by NI-DAQ, 1-7
libraries, NI-DAQ, 2-1
low-level data acquisition functions, 3-21 to 3-23
application tips. *See* data acquisition application tips.
DAQ_Check, 3-21
DAQ_Clear, 3-21
DAQ_Config, 3-21
DAQ_Monitor, 3-21
DAQ_Rate, 3-22
DAQ_Set_clock, 3-22
DAQ_Start, 3-22
DAQ_StopTrigger_Config, 3-22
DAQ_VScale, 3-22
Lab_ISCAN_Check, 3-22
Lab_ISCAN_Start, 3-22
SCAN_Demux, 3-22
SCAN_Sequence_Demux, 3-22 to 3-23, 3-33
SCAN_Sequence_Retrieve, 3-23, 3-33
SCAN_Sequence_Setup, 3-23, 3-33
SCAN_Setup, 3-23
SCAN_Start, 3-23
low-level double-buffered data acquisition functions, 3-23 to 3-24
application tips. *See* data acquisition application tips.
DAQ_DB_Config, 3-23, 4-10
DAQ_DB_HalfReady, 3-24, 4-12
DAQ_DB_Transfer, 3-24, 4-11
low-level waveform generation functions, 3-39 to 3-41
WFM_Chan_Control, 3-39, 3-52
WFM_Check, 3-39, 3-52
WFM_ClockRate, 3-39
WFM_DB_Config, 3-40
WFM_DB_HalfReady, 3-40
WFM_DB_Transfer, 3-40
WFM_Group_Control, 3-40
WFM_Group_Setup, 3-40
WFM_Load, 3-40

WFM_Rate, 3-40
WFM_Scale, 3-41
WFM_Set_Clock, 3-41
LPM devices. *See* PC-LPM-16.
LPM16_Calibrate function, 3-4

M

manual. *See* documentation.
Measurement & Automation Explorer application, 1-7
Microsoft Visual Basic for Windows
building Windows applications, 2-4 to 2-6
buffer allocation, 2-5 to 2-6
example programs, 2-5
parameter passing, 2-6
procedure, 2-4
string passing, 2-6
NI-DAQ events, 3-6 to 3-15
ActiveX controls, 3-6 to 3-7
Analog Alarm Event control, 3-11 to 3-13
Analog Trigger Event control, 3-9 to 3-11
General DAQ Event controls, 3-7 to 3-9
General DAQ Event example, 3-13 to 3-15
multiple controls, 3-13
using multiple controls, 3-13
special considerations, 2-5 to 2-6
Microsoft Visual C++
building Windows applications, 2-2 to 2-3
example programs, 2-2 to 2-3
special considerations, 2-3
MIO devices
counter usage in waveform generation, 3-50
FIFO lag effect, 3-51 to 3-52
MIO and AI device terminology (table), *xv-xvii*
reference voltages for waveform generation (table), 3-49
MIO_Config function, 3-4
MIO-E series devices. *See* E series devices.
Multiplexed mode applications, SCXI, 3-113 to 3-120
multiplexer device (AMUX-64T), 3-25
multirate scanning
flow chart for, 3-34
functions for, 3-33
purpose and use, 3-32 to 3-33

N

NI 4551 devices
counter usage, 3-50
FIFO lag effect, 3-51 to 3-52
NI Developer Zone, A-1
NI-DAQ events in Visual Basic for Windows, 3-6 to 3-15
ActiveX controls, 3-6 to 3-7
Analog Alarm Event control, 3-11 to 3-13
Analog Trigger Event control, 3-9 to 3-11
General DAQ Event controls, 3-7 to 3-9
General DAQ Event example, 3-13 to 3-15
multiple controls, 3-13
using multiple controls, 3-13
NI-DAQ installation. *See* installation.
NI-DAQ libraries for Windows, 2-1
NI-DAQ software
features, 1-1 to 1-2
hardware support (tables), 1-4 to 1-6
language support, 1-7
overview, 1-3
nidaq32.dll (note), 2-1
NI-TIO based devices. *See* clocks or time counters for NI-TIO devices.
no-handshaking mode, 3-54

O

one-shot analog input functions, 3-16 to 3-19
AI_Change_Parameter, 3-16
AI_Check, 3-16
AI_Clear, 3-16
AI_Configure, 3-16
AI_Mux_Config, 3-17
AI_Read, 3-17
AI_Read_Scan, 3-17
AI_Read_VScan, 3-17
AI_Setup, 3-17
AI_VRead, 3-17
AI_VScale, 3-17
application tips, 3-17 to 3-19
one-shot analog output functions
AO_Change_Parameter, 3-35
AO_Configure, 3-35
AO_Update, 3-35
AO_VScale, 3-36
AO_VWrite, 3-36
AO_Write, 3-36
application tips, 3-36 to 3-38
OUT0* signal (table), 3-104
OUT1* signal (table), 3-104
OUT2* signal (table), 3-104

P

Parallel mode applications, SCXI modules, 3-120 to 3-124
parameter passing in Windows applications
Microsoft Visual Basic, 2-6
Microsoft Visual C++, 2-3
pattern generation I/O with DIO-32F, 653*X,* PCI-6115, and PCI-6120 devices, 3-74 to 3-75
PC-DIO-24 groups of ports, 3-55
PC-DIO-96 groups of ports, 3-55
PCI-4451 devices
counter usage, 3-50
FIFO lag effect, 3-51 to 3-52
PCI-6115 and PCI-6120 devices
groups of ports, 3-56 to 3-57
pattern generation, 3-74 to 3-75
PCLK1 signal (table), 3-105
PCLK2 signal (table), 3-105
PC-LPM-16
counter/timer signals, 3-25
interval counter/timer operation, 3-92
LPM16_Calibrate function, 3-4
period and continuous pulse-width measurement applications, 3-89 to 3-91
plug-in device support
AT, PC and NEC buses (table), 1-4
PC Card, CardBus, and VXI buses (table), 1-5
PCI buses (table), 1-4
PXI buses (table), 1-5
programmable frequency output operation, 3-81
pulse generation
application tips, 3-83 to 3-84
CTR_Pulse function, 3-77
flow chart, 3-83
retriggerable one-shot pulse, 3-84
pulse per second synchronization, 3-96 to 3-97
pulse width measurement
event-counting application, 3-86 to 3-87
period and continuous pulse-width measurement applications, 3-89 to 3-91

R

reference voltages for analog output devices
bipolar output polarity (table), 3-49 to 3-50
unipolar output polarity (table), 3-49

REQ1 signal
653*X* RTSI connections (table), 3-105
DIO-32F RTSI connections (table), 3-104
REQ2 signal
653*X* RTSI connections (table), 3-105
DIO-32F RTSI connections (table), 3-104
retriggerable one-shot pulse, 3-84
RTD_Buf_Convert function, 5-2 to 5-4
RTD_Convert function
definition, 5-1
purpose and use, 5-2 to 5-4
RTSI bus
application tips, 3-106 to 3-107
description, 3-102 to 3-103
RTSI bus connections, 3-103 to 3-106
653*X* devices, 3-105 to 3-106
AT-AO-6/10, 3-104
DIO-32F, 3-104 to 3-105
E series, DSA, 660*XX,* and 671*X,* 3-103
RTSI bus trigger functions
RTSI_Clear, 3-102
RTSI_Clock, 3-102
RTSI_Conn, 3-102
RTSI_DisConn, 3-102
Select_Signal, 3-102

S

SCAN_Demux function, 3-22
SCAN_Op function, 3-21
SCAN_Sequence_Demux function, 3-22 to 3-23, 3-33
SCAN_Sequence_Retrieve function, 3-23, 3-33
SCAN_Sequence_Setup function, 3-23, 3-33
SCAN_Setup function, 3-23
SCAN_Start function, 3-23
SCAN_to_Disk function, 3-21
SCXI application tips, 3-112 to 3-124
analog output applications, 3-126
data acquisition rates, 3-124 to 3-126
digital applications, 3-127
general SCXIbus application flowchart, 3-112
Multiplexed mode, 3-113 to 3-120
Parallel mode, 3-120 to 3-124
SCXI-1200 module settling rates (table), 3-126
settling rates (table), 3-125
settling times (table), 3-125
SCXI functions, 3-107 to 3-111
SCXI_AO_Write, 3-107
SCXI_Cal_Constants, 3-107
SCXI_Calibrate, 3-107 to 3-108
SCXI_Calibrate_Setup, 3-108
SCXI_Change_Chan, 3-108
SCXI_Configure_Connection, 3-108
SCXI_Configure_Filter, 3-108
SCXI_Get_Chassis_Info, 3-109
SCXI_Get_Module_Info, 3-109
SCXI_Get_State, 3-109
SCXI_Get_Status, 3-109
SCXI_Load_Config, 3-109
SCXI_ModuleID_Read, 3-109
SCXI_MuxCtr_Setup, 3-110
SCXI_Reset, 3-110
SCXI_Scale, 3-110
SCXI_SCAN_Setup, 3-110
SCXI_Set_Config, 3-110
SCXI_Set_Excitation, 3-110
SCXI_Set_Gain, 3-111
SCXI_Set_Input_Mode, 3-111
SCXI_Set_State, 3-111
SCXI_Set_Threshold, 3-111
SCXI_Single_Chan_Setup, 3-111
SCXI_Track_Hold_Control, 3-111
SCXI_Track_Hold_Setup, 3-111
SCXI_Calibrate function, 3-4, 3-107 to 3-108
Select_Signal function, 3-4, 3-102
Set_DAQ_Device_Info function, 3-3
setting up DAQ systems. *See* device configuration; installation.

simultaneous counter operations (figure), 3-85
single-buffered versus double-buffered data, 4-1 to 4-2
single-channel analog input functions, 3-16 to 3-19
  AI_Change_Parameter, 3-16
  AI_Check, 3-16
  AI_Clear, 3-16
  AI_Configure, 3-16
  AI_Mux_Config, 3-17
  AI_Read, 3-17
  AI_Read_Scan, 3-17
  AI_Read_VScan, 3-17
  AI_Setup, 3-17
  AI_VRead, 3-17
  AI_VScale, 3-17
  application tips, 3-17 to 3-19
software-calibration and device-specific functions, 3-3 to 3-4
  AO_Calibrate, 3-3
  Calibrate_1200, 3-3 to 3-4
  Calibrate_DSA, 3-4
  Calibrate_E_Series, 3-4
  Calibrate_TIO, 3-4
  Configure_HW_Analog_Trigger, 3-4
  LPM16_Calibrate, 3-4
  MIO_Config, 3-4
  SCXI_Cal_Constants, 3-107
  SCXI_Calibrate, 3-4, 3-107 to 3-108
  SCXI_Calibrate_Setup, 3-108
  Select_Signal, 3-4, 3-102
square wave generation functions, 3-84
STOPTRIG1 signal (table), 3-105
STOPTRIG2 signal (table), 3-105
Strain_Buf_Convert function, 5-4 to 5-6
Strain_Convert function
  definition, 5-1
  purpose and use, 5-4 to 5-6
string passing in Windows applications
  Microsoft Visual Basic, 2-6
  Microsoft Visual C++, 2-3
synchronization of clocks. *See* clocks or time counters for NI-TIO devices.
system integration, by National Instruments, A-1

T

technical support resources, A-1 to A-2
Thermistor_Buf_Convert function, 5-7 to 5-9
Thermistor_Convert function
  definition, 5-1
  purpose and use, 5-7 to 5-9
Thermocouple_Buf_Convert function, 5-9 to 5-11
Thermocouple_Convert function
  definition, 5-1
  purpose and use, 5-9 to 5-11
time counters. *See* clocks or time counters for NI-TIO devices.
time-lapse measurement, 3-87
Timeout_Config function, 3-3
transducer conversion functions, 5-1 to 5-11
  overview, 5-1 to 5-2
  RTD_Buf_Convert, 5-2 to 5-4
  RTD_Convert, 5-1, 5-2 to 5-4
  Strain_Buf_Convert, 5-4 to 5-6
  Strain_Convert, 5-1, 5-4 to 5-6
  Thermistor_Buf_Convert, 5-7 to 5-9
  Thermistor_Convert, 5-1, 5-7 to 5-9
  Thermocouple_Buf_Convert, 5-9 to 5-11
  Thermocouple_Convert, 5-1, 5-9 to 5-11

V

Visual Basic. *See* Microsoft Visual Basic for Windows.

W

waveform generation application
tips, 3-41 to 3-52
basic applications, 3-41 to 3-43
basic waveform generation with pauses, 3-44 to 3-46
counter usage, 3-50 to 3-51
double-buffered applications, 3-47 to 3-48
external triggering, 3-52
FIFO lag effect, 3-51 to 3-52
minimum update intervals, 3-50
reference voltages for analog output devices
bipolar output polarity (table), 3-49 to 3-50
unipolar output polarity (table), 3-49
waveform generation functions, 3-39 to 3-41
high-level waveform generation functions, 3-39
WFM_from_Disk, 3-39
WFM_Op, 3-39
low-level waveform generation functions, 3-39 to 3-41
WFM_Chan_Control, 3-39, 3-52
WFM_Check, 3-39, 3-52
WFM_ClockRate, 3-39
WFM_DB_Config, 3-40, 4-10
WFM_DB_HalfReady, 3-40, 4-12
WFM_DB_Transfer, 3-40, 4-11
WFM_Group_Control, 3-40
WFM_Group_Setup, 3-40
WFM_Load, 3-40
WFM_Rate, 3-40
WFM_Scale, 3-41
WFM_Set_Clock, 3-41
Web support from National Instruments, A-1
WFM_Chan_Control function, 3-39, 3-52
WFM_Check function, 3-39, 3-52
WFM_ClockRate function, 3-39
WFM_DB_Config function, 3-40, 4-10
WFM_DB_HalfReady function, 3-40, 4-12
WFM_DB_Transfer function, 3-40, 4-11
WFM_from_Disk function, 3-39
WFM_Group_Control function, 3-40
WFM_Group_Setup function, 3-40
WFM_Load function, 3-40
WFM_Op function, 3-39
WFM_Rate function, 3-40
WFM_Scale function, 3-41
WFM_Set_Clock function, 3-41
Windows applications, building, 2-1 to 2-10
Borland C++, 2-7 to 2-8
Borland Delphi, 2-8
Microsoft Visual Basic, 2-4 to 2-6
Microsoft Visual C++, 2-2 to 2-3
NI-DAQ examples, 2-9 to 2-10
NI-DAQ libraries, 2-1
Worldwide technical support, A-2