LabVIEW™

Measurements Manual

NATIONAL INSTRUMENTS™

April 2003 Edition
Part Number 322661B-01

Important Information

Contents

Chapter 4
Measurement Fundamentals

Chapter 5
Creating a Typical Measurement Application

About This Manual

The *LabVIEW Measurements Manual* contains information you need to acquire and analyze measurement data in LabVIEW. You should have a basic knowledge of LabVIEW before you read this manual. If you have never used LabVIEW, refer to the Getting Started with LabVIEW manual.

This manual supplements the *LabVIEW User Manual* and assumes that you are familiar with that material. You also should be familiar with the operation of LabVIEW, the operating systems, and your data acquisition (DAQ) device.

Note LabVIEW Real-Time applications require special consideration. Refer to the *LabVIEW Real-Time Module User Manual* for more information about creating real-time applications.

Conventions

The following conventions appear in this manual:

[ ]	Square brackets enclose optional items—for example, [`response`].
»	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 tip, which alerts you to advisory information.
📝	This icon denotes a note, which alerts you to important information.
bold	Bold text denotes items that you must select or click in the software, such as menu items and dialog box options. Bold text also denotes parameter names.
italic	Italic text denotes variables, emphasis, or a cross reference. This font also denotes text that is a placeholder for a word or value that you must supply.
`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, variables, filenames and extensions, and code excerpts.

Platform	Text in this font denotes a specific platform and indicates that the text following it applies only to that platform.

Related Documentation

The following documents and online help contain information that you might find helpful as you read this manual:

- *Getting Started with LabVIEW*
- *LabVIEW User Manual*
- *LabVIEW Help*
- *LabVIEW Analysis Concepts*
- *Measurement & Automation Explorer Help*
- *DAQ Quick Start Guide for NI-DAQ 7.0*
- *Data Acquisition VIs for Traditional NI-DAQ* help
- *DAQmx Data Acquisition VIs* help
- *NI-DAQmx Help*
- *Getting Started with SCXI*
- *LabVIEW Real-Time Module User Manual*
- *LabVIEW Real-Time Module for Mac OS X User Manual Addendum*
- NI Developer Zone at `ni.com/zone`

1 Overview of Measurement and Virtual Instrumentation

Taking measurements with instruments helps science and technology progress. Scientists and engineers use instruments to observe, control, and understand the physical universe, to perform research in life sciences and medicine; to design, test, and manufacture electronics; and to improve machine and process control in countless industries.

To understand how instruments are built, consider the history of instrumentation. Instruments have always made use of widely available technology. In the 19th century, the jeweled movement of the clock was first used to build analog meters. In the 1930s, the variable capacitor, the variable resistor, and the vacuum tube from radios were used to build the first electronic instruments. Display technology from the television has contributed to modern oscilloscopes and analyzers. And finally, modern personal computers contribute high-performance computation and display capabilities at an ever-improving performance-to-price ratio.

Virtual Instrumentation

Virtual instrumentation combines hardware and software with industry-standard computer technologies to create user-defined instrumentation solutions. National Instruments specializes in developing plug-in and distributed hardware and driver software for data acquisition (DAQ), IEEE 488 (GPIB), PXI, serial, and industrial communications. The driver software is the application programming interface to the hardware and is consistent across National Instruments application software, such as LabVIEW, LabWindows™/CVI™, and Measurement Studio. These platforms deliver the sophisticated display and analysis capabilities that virtual instrumentation requires.

You can use virtual instrumentation to create a complete and customized system for test, measurement, and industrial automation by combining different hardware and software components. If the system changes, you often can reuse the virtual instrument components without purchasing additional hardware or software.

System Components for Taking Measurements with Virtual Instruments

Different hardware and software components can make up a virtual instrumentation system. This manual describes many of these options. You can use a variety of hardware components to monitor or control a process or to test a device. As long as you can connect the hardware to the computer and understand how the hardware takes measurements, you can incorporate hardware into a virtual instrumentation system.

2

Comparing DAQ Devices and Computer-Based Instruments for Data Acquisition

The fundamental task of all measurement systems is the measurement and/or generation of real-world physical signals. Measurement devices help you acquire, analyze, and present the measurements you take.

Through data acquisition, you acquire and convert physical signals, such as voltage, current, pressure, and temperature, into digital formats and transfer them into the computer. Popular methods for acquiring data include plug-in DAQ and instrument devices, GPIB instruments, PXI (PCI eXtensions for Instrumentation) instruments, and RS-232 instruments.

Through data analysis, you transform raw data into meaningful information by using curve fitting, statistical analysis, frequency response, or other numerical operations.

Through data presentation, you display data in a graph, thermometer, table, or other visual display.

Building a computer-based measurement system can be a daunting task. This chapter describes several types of hardware solutions to help you determine which hardware components to use in the measurements systems you build.

General-Purpose DAQ

A general-purpose DAQ device is a device that acquires or generates data and can contain multiple channels. You also can use general-purpose DAQ devices to generate analog signals, such as a sine wave, and digital signals, such as a pulse. Typically, you connect these devices directly to the internal bus of a computer through a plug-in slot.

A general-purpose DAQ measurement system is different from other measurement systems because the software installed on a computer performs the actual measurements. The DAQ device only converts the incoming signal into a digital signal the computer can use. This means that the same DAQ device can perform a multitude of measurements simply by changing the software application that reads the data. In addition to acquiring the data, the application for a DAQ measurement system also uses the software that processes the data and displays the results. Although this flexibility allows you to have one hardware device for many types of measurements, you must spend more time developing the different applications for the different types of measurements. LabVIEW includes many acquisition and analysis functions to help you develop different applications.

How Computers Talk to DAQ Devices

Before a computer-based measurement system can measure a physical signal, such as temperature, a sensor or transducer must convert the physical signal into an electrical one, such as voltage or current. You might consider the plug-in DAQ device to be the entire measurement system, but it is actually only one system component. You cannot always directly connect signals to a plug-in DAQ device. In these cases, you must use signal conditioning accessories to condition the signals before the plug-in DAQ device converts them to digital information. The software controls the DAQ system by acquiring the raw data, analyzing, and presenting the results.

Consider the following options for a DAQ system:

- The plug-in DAQ device resides in the computer. You can plug the device into the PCI slot of a desktop computer or the PCMCIA slot of a laptop computer for a portable DAQ measurement system.
- The DAQ device is external and connects to the computer through an existing port, such as the serial port or Ethernet port, which means you can quickly and easily place measurement nodes near sensors.

Role of Software

The computer receives raw data through the DAQ device. The application you write presents and manipulates the raw data in a form you can understand. The software also controls the DAQ system by commanding the DAQ device when and from which channels to acquire data.

Typically, DAQ software includes drivers and application software. Drivers are unique to the device or type of device and include the set of commands the device accepts. Application software, such as LabVIEW, sends the drivers commands, such as acquire and return a thermocouple reading. The application software also displays and analyzes the acquired data.

NI measurement devices include NI-DAQ driver software, a collection of VIs you use to configure, acquire data from, and send data to the measurement devices.

NI-DAQ

NI-DAQ 7.0 contains two NI-DAQ drivers—Traditional NI-DAQ and NI-DAQmx—each with its own application programming interface (API), hardware configuration, and software configuration.

- Traditional NI-DAQ is an upgrade to NI-DAQ 6.9.*x*, the earlier version of NI-DAQ. Traditional NI-DAQ has the same VIs and functions and works the same way as NI-DAQ 6.9.*x*. You can use Traditional NI-DAQ on the same computer as NI-DAQmx, which you cannot do with NI-DAQ 6.9.*x*.
- NI-DAQmx is the latest NI-DAQ driver with new VIs, functions, and development tools for controlling measurement devices. The advantages of NI-DAQmx over previous versions of NI-DAQ include the DAQ Assistant for configuring channels and measurement tasks for a device; increased performance, including faster single-point analog I/O and multithreading; and a simpler API for creating DAQ applications using fewer functions and VIs than earlier versions of NI-DAQ.

Traditional NI-DAQ and NI-DAQmx support different sets of devices. Refer to the National Instruments Web site at `ni.com/daq` for the list of supported devices.

Refer to the *DAQ Quick Start Guide for NI-DAQ 7.0* for more information about Traditional NI-DAQ and NI-DAQmx and to determine which to use.

Instrument I/O

Many instruments are external to the computer and do not rely on a computer to take a measurement. By connecting instruments to a computer, you can programatically control and monitor the instruments and collect data that you can process further or store in files. You can install some instruments in a computer similar to general-purpose DAQ devices. These internal instruments are called modular instruments.

Regardless of how you connect to an instrument, the computer must use a specific protocol to communicate with the instrument. How the computer controls the instrument and acquires data from the instrument depends on the type of the instrument. GPIB, serial port, and PXI are common types of instruments.

Like general-purpose DAQ devices, instruments digitize data, but they have a special purpose or are designed for a specific type of measurement. For standalone instruments, you generally cannot modify the software that processes the data and calculates the result because the software usually is built into the instrument.

Because modular instrumentation uses software running on standard PC technology, you can more easily modify the behavior of these instruments. For example, with some digital multimeter modular instruments, you can program the instruments to acquire a buffer of data at a high rate of speed, much like an oscilloscope.

How Computers Control Instruments

Computers control instruments by sending commands to the instruments over a bus, such as GPIB, PXI, or RS-232. For example, you can send a command to the instrument to measure a signal and then send another command to send the resulting data over the bus back to the computer.

Instrument Drivers

An instrument driver is a collection of functions that control and operate the instrument. The instrument drivers simplify instrument programming to high-level commands so you do not need to learn low-level, instrument-specific syntax. Instrument drivers are not required to use an instrument, but are designed to help save you time as you develop an application.

Using instrument drivers provides the following advantages:

- You can use instrument drivers to quickly build complete systems. Instrument drivers receive, parse, and scale the response strings from instruments into scaled data that you can use in test programs.
- Instrument drivers can reduce software development costs because developers do not need to spend time understanding low-level command syntax needed to program instruments.
- Instrument drivers can help make test programs easier to maintain because instrument drivers contain all the I/O for an instrument within one library, separate from other code, which is easy to upgrade when or if the hardware changes.

NI provides more than 2,200 instrument drivers from more than 150 vendors. Refer to the NI Instrument Driver Network at `ni.com/idnet` for a list of available instrument drivers.

3 Configuring Measurement Hardware

This chapter includes installation and configuration information for National Instruments measurement devices.

The NI-DAQ driver software provides LabVIEW with a high-level interface to National Instruments DAQ devices. Driver software, such as NI-DMM, NI-SCOPE, NI-FGEN, and NI-SWITCH, provides LabVIEW with high-level interfaces to modular instrumentation. Use driver software such as NI-488.2, NI-VISA, and Interchangeable Virtual Instruments (IVI) to communicate with standalone instruments.

The LabVIEW VIs call into the driver software, which communicates with the measurement devices.

Installing and Configuring Hardware

Before you begin developing measurement applications, you must install and configure the measurement hardware. The software drivers need the hardware configuration information to program the hardware properly.

Each system architecture is different. Some systems might use general-purpose plug-in DAQ devices. Others systems might use special-purpose instruments controlled through GPIB, serial, or Ethernet. Each system requires a unique configuration procedure to ensure the measurement devices work properly and can coexist with other peripherals. However, in most cases, you can complete the following steps to install a measurement device.

1. Install LabVIEW and the driver software. The LabVIEW installer installs National Instruments driver software if the version included with LabVIEW is newer than any previously installed version of the driver software. To ensure you install a version that supports the device, install it from the CD packaged with the device. Refer to the *DAQ Quick Start Guide for NI-DAQ 7.0* for more information about installing NI-DAQ.

Note For Windows 2000/NT/XP Professional, log on as an administrator when you install LabVIEW and the driver software and when you configure the measurement hardware.

2. Power off the computer.
3. Install the measurement hardware. Before you install the measurement hardware, refer to the device documentation to see if you need to change any hardware-selectable options. For example, some devices have jumpers to select analog input polarity, input mode, analog output reference, and so on. Make a note of which options you change so you can notify the driver software by entering the information in one of the configuration utilities or by using VI or function calls in the application.
4. Power on the computer.
5. Configure the measurement hardware using **(Windows)** Measurement & Automation Explorer (MAX) or **(Macintosh)** the Configuration Utility.

Refer to the device documentation, the *Measurement & Automation Explorer Help*, or the Troubleshooting Wizards available at `ni.com/support` for more information about installing and configuring measurement hardware.

Configuring Hardware on Windows

After you install a measurement device, use the following utilities to configure the hardware on Windows operating systems.

Measurement & Automation Explorer

MAX is a Windows-based application installed during the National Instruments driver software installation. Use MAX to configure NI software and hardware, execute system diagnostics, add new channels and interfaces, and view the devices and instruments you have connected. You must use MAX to configure devices when programming with Traditional NI-DAQ. Double-click the **Measurement & Automation** icon on the desktop to launch MAX.

DAQ Assistant

The DAQ Assistant is a graphical interface for configuring NI-DAQmx measurement tasks, channels, and scales for use in LabVIEW 7.0 and later. Use the DAQ Assistant to generate NI-DAQmx code to run tasks and channels, or to deploy the NI-DAQmx code to another DAQ system. Use LabVIEW or MAX to launch the DAQ Assistant.

Configuring VISA Devices and IVI Logical Names

You can assign meaningful Virtual Instrument Software Architecture (VISA) aliases and IVI logical names to the instruments you control using VISA and IVI. Assign VISA aliases in the **Devices and Interfaces** category in MAX. Configure IVI logical names in the **IVI** category in MAX. You can use the aliases and logical names in LabVIEW application development to refer to an instrument. For example, you can assign the alias `scope` to refer to the serial port or GPIB address for an oscilloscope.

Configuring FieldPoint Modules

You can configure FieldPoint communication resources, modules, and items in MAX. Configure the communication resources and modules in the **Devices and Interfaces** category if you are using a serial device. If you are using an Ethernet device, use the **Remote Systems** category. Configure items in the **Data Neighborhood** category. Use the communications resource name, module name, and item name when you perform I/O in LabVIEW.

Configuring Hardware on Mac OS

After you install a measurement device, use the following utilities to configure the hardware on Mac OS 9 and earlier.

Note DAQ devices are supported on Mac OS X through real-time targets. Refer to the *LabVIEW Real-Time Module for Mac OS X User Manual Addendum* for more information about configuring DAQ devices on Mac OS X.

NI-DAQ Configuration Utility

Use the NI-DAQ Configuration Utility to configure the parameters for a DAQ device installed in a Macintosh computer. Mac OS automatically recognizes DAQ devices. After you install a DAQ device, you must use the NI-DAQ Configuration Utility to assign a device number to the device. The configuration utility saves the device number and the configuration parameters for the DAQ device and SCXI (Signal Conditioning eXtensions for Instrumentation) system. After you configure the DAQ device, you do not need to run the NI-DAQ Configuration Utility again unless you change the system parameters. The LabVIEW folder contains a shortcut to the NI-DAQ Configuration Utility.

NI-488.2 Configuration Utility

Use the NI-488.2 Configuration Utility to configure the parameters for the GPIB devices installed in a Macintosh computer. Mac OS automatically recognizes GPIB devices. You can use the NI-488.2 Configuration Utility to view or modify the default configuration settings.

Configuring Serial Ports on Macintosh

Use the VISA Find Resource function to automatically detect new ports and to assign VISA resource names.

4 Measurement Fundamentals

This chapter introduces you to concepts you should be familiar with for taking measurements with DAQ devices and instruments.

Signal Acquisition

Signal acquisition is the process of converting physical phenomena into data the computer can use. A measurement starts with using a transducer to convert a physical phenomenon into an electrical signal. Transducers can generate electrical signals to measure such things as temperature, force, sound, or light. Table 4-1 lists some common transducers.

Table 4-1. Phenomena and Transducers

Phenomena	Transducer
Temperature	Thermocouples Resistance temperature detectors (RTDs) Thermistors Integrated circuit sensors
Light	Vacuum tube photosensors Photoconductive cells
Sound	Microphones
Force and pressure	Strain gages Piezoelectric transducers Load cells
Position (displacement)	Potentiometers Linear voltage differential transformers (LVDT) Optical encoders
Fluid flow	Head meters Rotational flowmeters Ultrasonic flowmeters
pH	pH electrodes

Signal Sources

Analog input acquisitions use grounded and floating signal sources.

Grounded Signal Sources

A grounded signal source is one in which the voltage signals are referenced to a system ground, such as the earth or a building ground, as shown in Figure 4-1. Because such sources use the system ground, they share a common ground with the measurement device. The most common examples of grounded sources are devices that plug into a building ground through wall outlets, such as signal generators and power supplies.

Note The grounds of two independently grounded signal sources generally are not at the same potential. The difference in ground potential between two instruments connected to the same building ground system is typically 10 mV to 200 mV. The difference can be higher if power distribution circuits are not properly connected. This causes a phenomenon known as a ground loop.

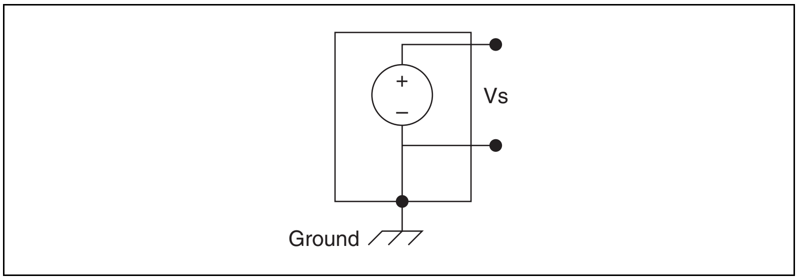


Figure 4-1. Grounded Signal Source

Floating Signal Sources

In a floating signal source, the voltage signal is not referenced to any common ground, such as the earth or a building ground, as shown in Figure 4-2. Some common examples of floating signal sources are batteries, thermocouples, transformers, and isolation amplifiers. Notice in Figure 4-2 that neither terminal of the source is connected to the electrical outlet ground as in Figure 4-1. Each terminal is independent of the system ground.

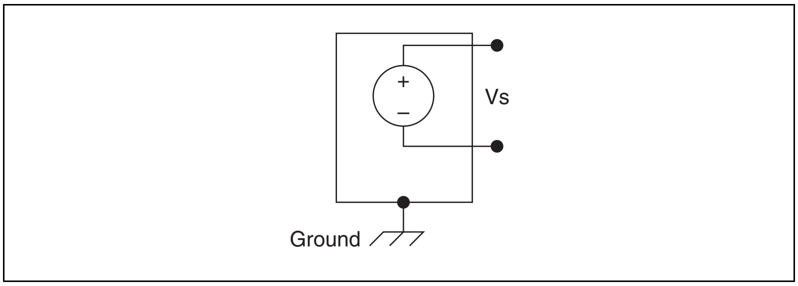


Figure 4-2. Floating Signal Source

Signal Conditioning

Signal conditioning is the process of measuring and manipulating signals to improve accuracy, isolation, filtering, and so on.

To measure signals from transducers, you must convert them into a form a DAQ device can accept. For example, the output voltage of most thermocouples is very small and susceptible to noise. Therefore, you might need to amplify the thermocouple output before you digitize it. This amplification is a form of signal conditioning. Common types of signal conditioning include amplification, linearization, transducer excitation, and isolation.

Figure 4-3 shows some common types of transducers and signals and the signal conditioning each requires.

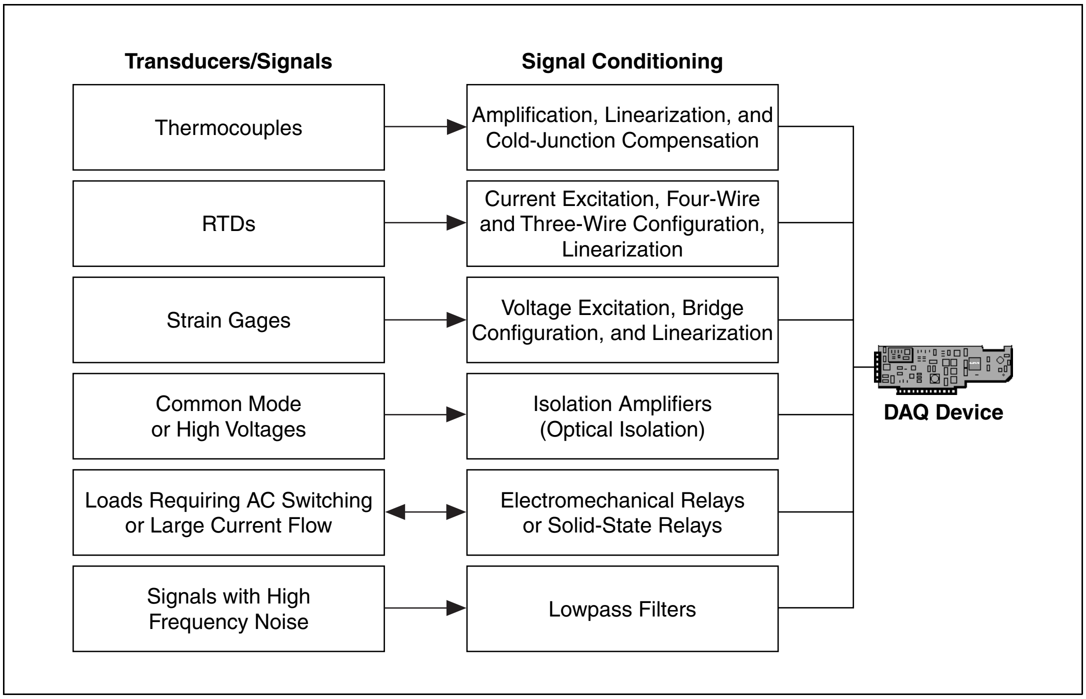

Transducers/Signals
Signal Conditioning
Thermocouples
Amplification, Linearization, and Cold-Junction Compensation
RTDs
Current Excitation, Four-Wire and Three-Wire Configuration, Linearization
Strain Gages
Voltage Excitation, Bridge Configuration, and Linearization
Common Mode or High Voltages
Isolation Amplifiers (Optical Isolation)
Loads Requiring AC Switching or Large Current Flow
Electromechanical Relays or Solid-State Relays
Signals with High Frequency Noise
Lowpass Filters
DAQ Device

Figure 4-3. Common Types of Transducers/Signals and Signal Conditioning

Amplification

Amplification is the most common type of signal conditioning. Amplifying electrical signals improves accuracy in the resulting digitized signal and reduces the effects of noise.

Amplify low-level signals at the DAQ device or SCXI module (labeled as the External Amplifier in Figure 4-4) located nearest to the signal source to increase the signal-to-noise ratio (SNR). For the highest possible accuracy, amplify the signal so the maximum voltage range equals the maximum input range of the analog-to-digital converter (ADC).

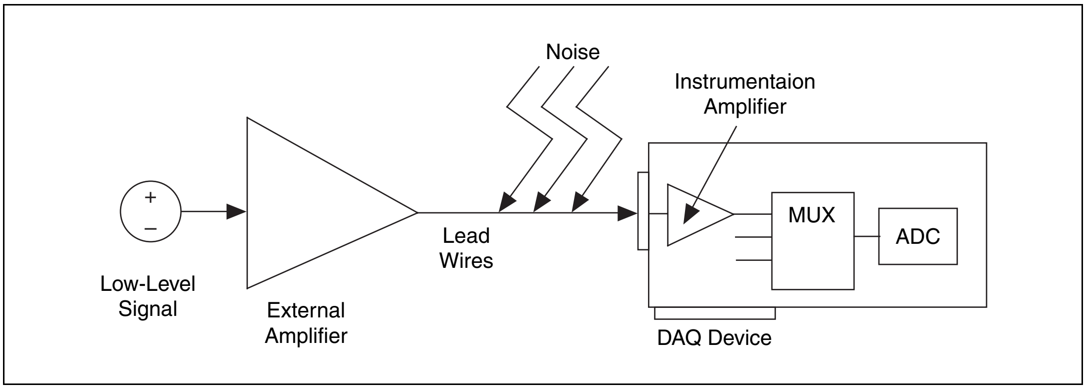


Figure 4-4. Amplifying Signals near the Source to Increase Signal-to-Noise Ratio (SNR)

If you amplify the signal at the DAQ device, the signal is measured and digitized with noise that might have entered the lead wire, which decreases SNR. However, if you amplify the signal close to the signal source with a SCXI module, noise has a less destructive effect on the signal, and the digitized representation is a better reflection of the original low-level signal. Refer to the National Instruments Web site at `ni.com/info` and enter the info code `exd2hc` for more information about analog signals.

Tip There are several ways to reduce noise:

- Use shielded cables or a twisted pair of cables.
- Minimize wire length to minimize noise the lead wires pick up.
- Keep signal wires away from AC power cables and monitors to reduce 50 or 60 Hz noise.

Linearization

Many transducers, such as thermocouples, have a nonlinear response to changes in the physical phenomena you measure. LabVIEW can linearize the voltage levels from transducers so you can scale the voltages to the measured phenomena. LabVIEW provides scaling functions to convert voltages from strain gages, RTDs, thermocouples, and thermistors.

Transducer Excitation

Signal conditioning systems can generate excitation, which some transducers require for operation. Strain gages and RTDs require external voltage and currents, respectively, to excite their circuitry into measuring physical phenomena. This type of excitation is similar to a radio that needs power to receive and decode audio signals.

Isolation

Another common way to use signal conditioning is to isolate the transducer signals from the computer for safety purposes. When the signal you monitor contains large voltage spikes that could damage the computer or harm the operator, do not connect the signal directly to a DAQ device without some type of isolation.

You also can use isolation to ensure that differences in ground potentials do not affect measurements from the DAQ device. When you do not reference the DAQ device and the signal to the same ground potential, a ground loop can occur. Ground loops can cause an inaccurate representation of the measured signal. If the potential difference between the signal ground and the DAQ device ground is large, damage can occur to the measuring system. Using isolated SCXI modules eliminates the ground loop and ensures that the signals are accurately measured.

Measurement Systems

You configure a measurement system based on the hardware you use and the measurement you take.

Differential Measurement Systems

Differential measurement systems are similar to floating signal sources in that you make the measurement with respect to a floating ground that is different from the measurement system ground. Neither of the inputs of a differential measurement system are tied to a fixed reference, such as the earth or a building ground. Handheld, battery-powered instruments and DAQ devices with instrumentation amplifiers are examples of differential measurement systems.

A typical National Instruments device uses an implementation of an eight-channel differential measurement systems as shown in Figure 4-5. Using analog multiplexers in the signal path increases the number of measurement channels when only one instrumentation amplifier exists.

In Figure 4-5, the AIGND (analog input ground) pin is the measurement system ground.

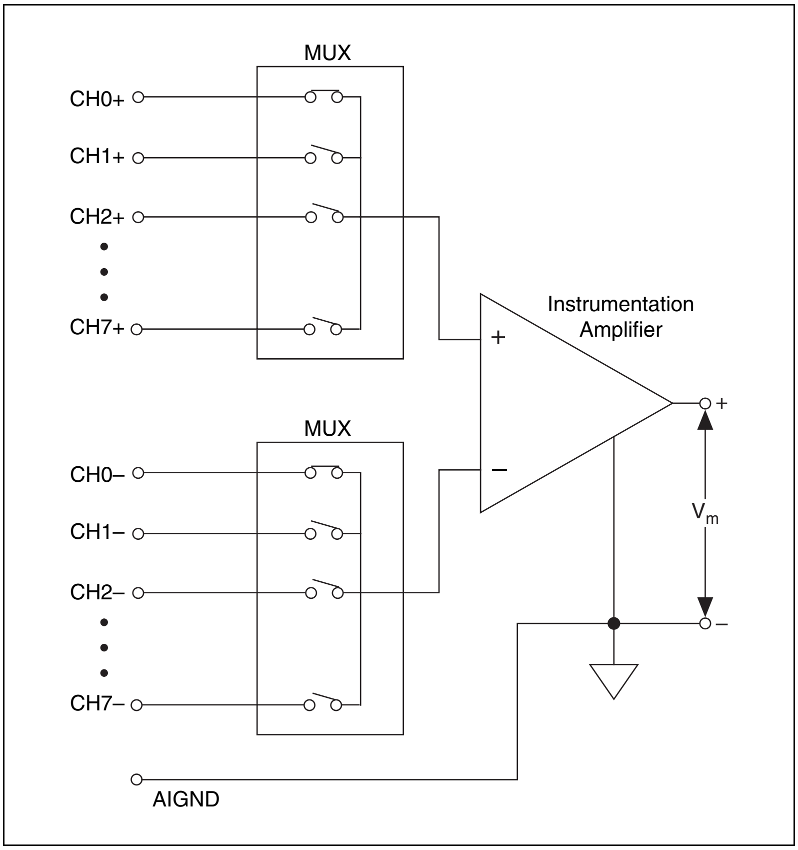


Figure 4-5. Differential Measurement System

Common Mode

An ideal differential measurement system responds only to the potential difference between its two terminals—the positive (+) and negative (–) inputs. A common-mode voltage is any voltage you measure with respect to the instrumentation amplifier ground present at both amplifier inputs. An ideal differential measurement system completely rejects, or does not measure, common-mode voltage. Rejecting common-mode voltage is useful because unwanted noise often is introduced as common-mode voltage in the circuit that makes up the cabling system of a measurement system.

However, several factors, such as the common-mode voltage range and the common-mode rejection ratio (CMRR) parameters, limit the ability of practical, real-world differential measurement systems to reject the common-mode voltage.

Common-Mode Voltage

The common-mode voltage range limits the allowable voltage range on each input with respect to the measurement system ground. Violating this constraint results not only in measurement error but also in possible damage to components on the device. The following equation defines common-mode voltage (V_{cm}):

$$V_{cm} = \frac{(V_{+} + V-)}{2}$$

where V_{+} is the voltage at the noninverting terminal of the measurement system with respect to the measurement system ground, and $V-$ is the voltage at the inverting terminal of the measurement system with respect to the measurement system ground.

Common-Mode Rejection Ratio

CMRR measures the ability of a differential measurement system to reject the common-mode voltage signal. The CMRR is a function of frequency and typically reduces with frequency. The higher the CMRR, the better the amplifier can extract differential signals in the presence of common-mode noise. Using a balanced circuit can optimize the CMRR. Most DAQ devices specify the CMRR up to the power line frequency, which is 60 Hz. The following equation defines CMRR in decibels (dB):

$$\text{CMRR(db)} = 20\log\left(\frac{\text{Differential Gain}}{\text{Common-Mode Gain}}\right)$$

Figure 4-6 shows a simple circuit in which CMRR in decibels is measured as:

$$20\log\frac{V_{\text{cm}}}{V_{\text{out}}}$$

where $V_{+} + V- = V_{\text{cm}}$

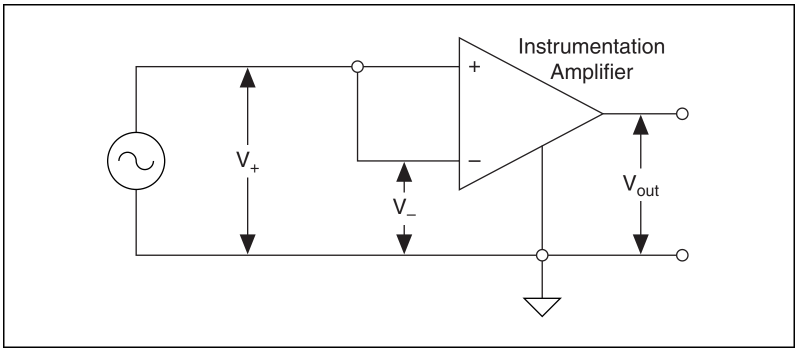


Figure 4-6. Common-Mode Rejection Ratio (CMRR)

Referenced and Non-Referenced Single-Ended Measurement Systems

Referenced and non-referenced single-ended measurement systems are similar to grounded sources in that you make the measurement with respect to a ground. A referenced single-ended measurement system measures voltage with respect to the ground, AIGND, which is directly connected to the measurement system ground. Figure 4-7 shows a 16-channel referenced single-ended measurement system.

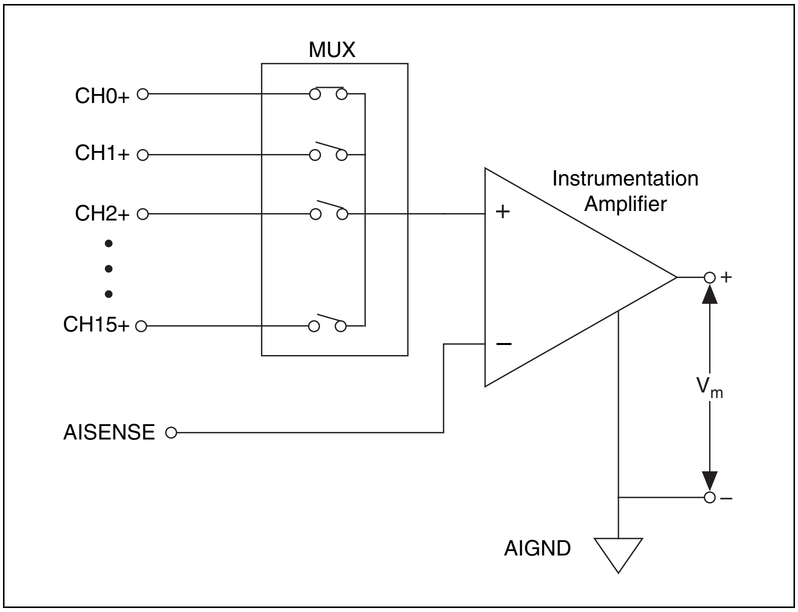


Figure 4-7. Referenced Single-Ended Measurement System

DAQ devices often use a non-referenced single-ended (NRSE) measurement technique, or pseudodifferential measurement, which is a variant of the referenced single-ended measurement technique. Figure 4-8 shows a NRSE system.

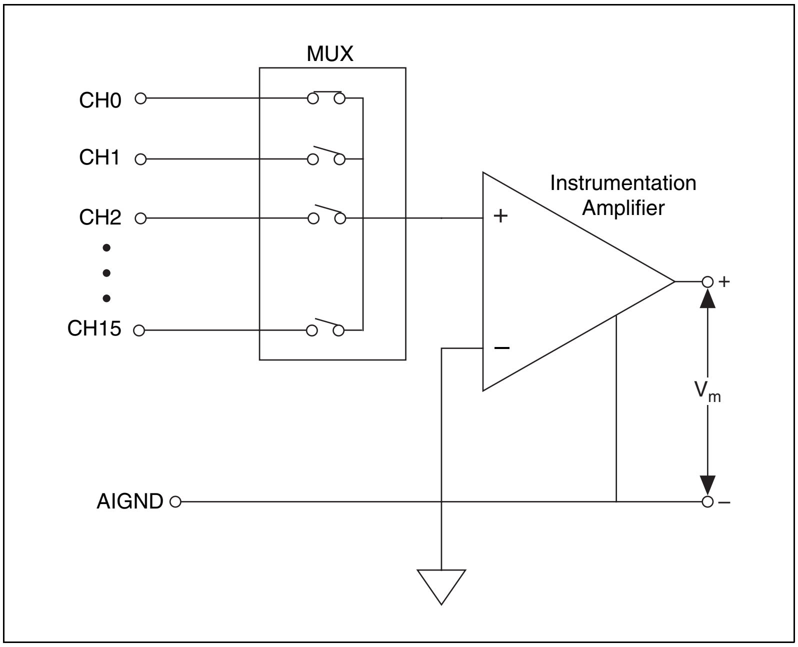


Figure 4-8. NRSE Measurement System

In a NRSE measurement system, all measurements are still made with respect to a single-node analog input sense (AISENSE on E Series devices), but the potential at this node can vary with respect to the measurement system ground (AIGND). A single-channel NRSE measurement system is the same as a single-channel differential measurement system.

Summary of Signal Sources and Measurement Systems

Figure 4-9 summarizes ways to connect a signal source to a measurement system.

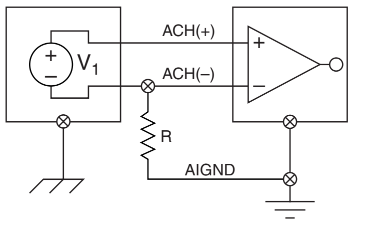
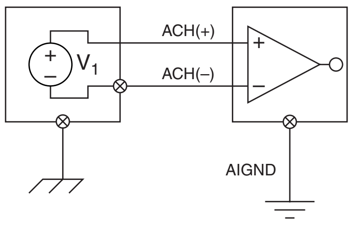
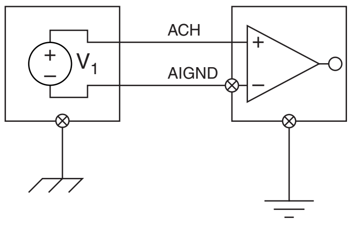
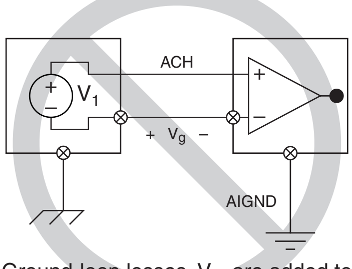
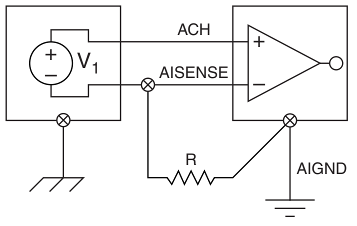
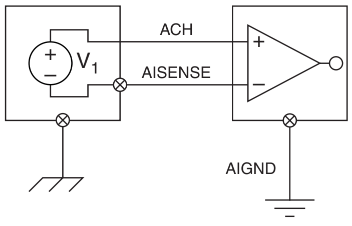
Input	**Signal Source Type**	
	Floating Signal Source (Not Connected to Building Ground)	**Grounded Signal Source**
	Examples • Ungrounded Thermocouples • Signal Conditioning with Isolated Outputs • Battery Devices	Examples • Plug-in Instruments with Nonisolated Outputs
Differential (DIFF)	 See text for information on bias resistors.	
Single-Ended — Ground Referenced (RSE)		NOT RECOMMENDED  Ground-loop losses, V_g, are added to measured signal.
Single-Ended — Nonreferenced (NRSE)	 See text for information on bias resistors.	

Figure 4-9. Connecting a Signal Source to a Measurement System

Hardware versus Software Timing

You can use hardware timing or software timing to control when to acquire or generate a signal. With hardware timing, a clock on the device controls the rate. With software timing, the software, not the measurement device, determines the rate at which to acquire or generate samples. A hardware clock can run much faster and is more accurate than a software loop.

Note Some devices do not support hardware timing. Consult the device documentation to determine if a device supports hardware timing.

Sampling Rate

One of the most important elements of an analog input or analog output measurement system is the rate at which the measurement device samples an incoming signal or generates the output signal. The scan rate, or the sampling rate in NI-DAQmx, determines how often an analog-to-digital (A/D) or digital-to-analog (D/A) conversion takes place. A fast input sampling rate acquires more points in a given time and can form a better representation of the original signal than a slow input sampling rate can.

Aliasing

Sampling too slowly results in aliasing, which is a misrepresentation of the analog signal. Undersampling causes the signal to appear as if it has a different frequency than it actually does. To avoid aliasing, sample several times faster than the frequency of the signal.

For frequency measurements, according to the Nyquist theorem, you must sample at a rate greater than twice the maximum frequency component in the signal you are acquiring to accurately represent the signal. The Nyquist frequency is the maximum frequency you can represent without aliasing for a given sampling rate. The Nyquist frequency is one half the sampling frequency. Signals with frequency components above the Nyquist frequency appear aliased between DC and the Nyquist frequency. The alias frequency is the absolute value of the difference between the frequency of the input signal and the closest integer multiple of the sampling rate.

For example, assume the sampling frequency, *fs*, is 100 Hz. Also assume that the input signal contains the following frequencies: 25 Hz, 70 Hz, 160 Hz, and 510 Hz, as shown in Figure 4-10.

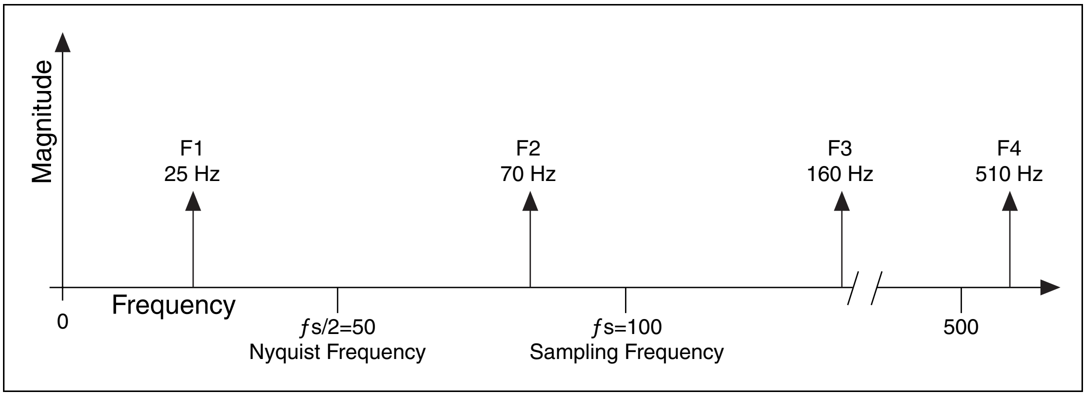


Figure 4-10. Non-Aliased Nyquist Frequency

Frequencies below the Nyquist frequency (*fs*/2 = 50 Hz) are sampled correctly, as shown in Figure 4-11. Frequencies above the Nyquist frequency appear as aliases. For example, F_1 (25 Hz) appears at the correct frequency, but F_2 (70 Hz), F_3 (160 Hz), and F_4 (510 Hz) have aliases at 30 Hz, 40 Hz, and 10 Hz, respectively.

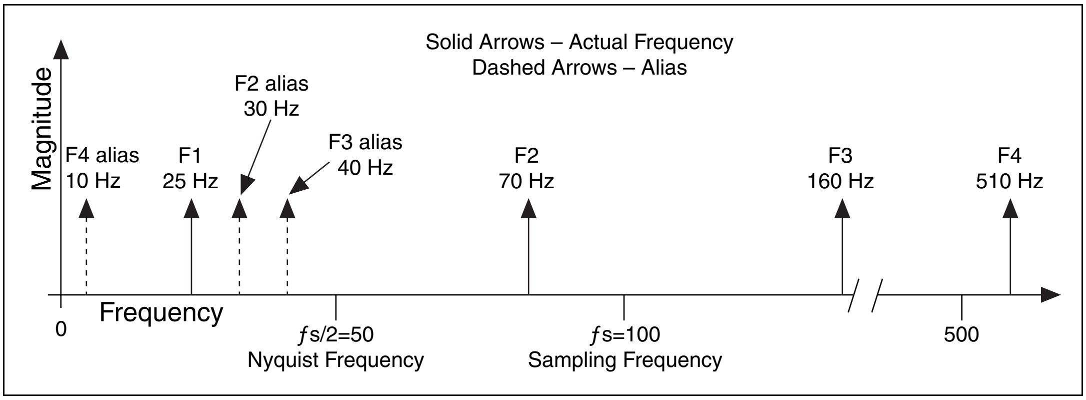


Figure 4-11. Aliased Nyquist Frequency Example

Use the following equation to calculate the alias frequency:

$$\text{Alias Freq.} = \text{ABS}(\text{Closest Int. Mult. of Sampling Freq.} - \text{Input Freq.})$$

where ABS means the absolute value. For example,

$$\text{Alias } F_2 = |100 - 70| = 30 \text{ Hz}$$

$$\text{Alias } F_3 = |(2)100 - 160| = 40 \text{ Hz}$$

$$\text{Alias } F_4 = |(5)100 - 510| = 10 \text{ Hz}$$

Determining How Fast to Sample

You might want to sample at the maximum rate available on the measurement device. However, if you sample very fast over long periods of time, you might not have enough memory or hard disk space to hold the data. Figure 4-12 shows the effects of various sampling rates.

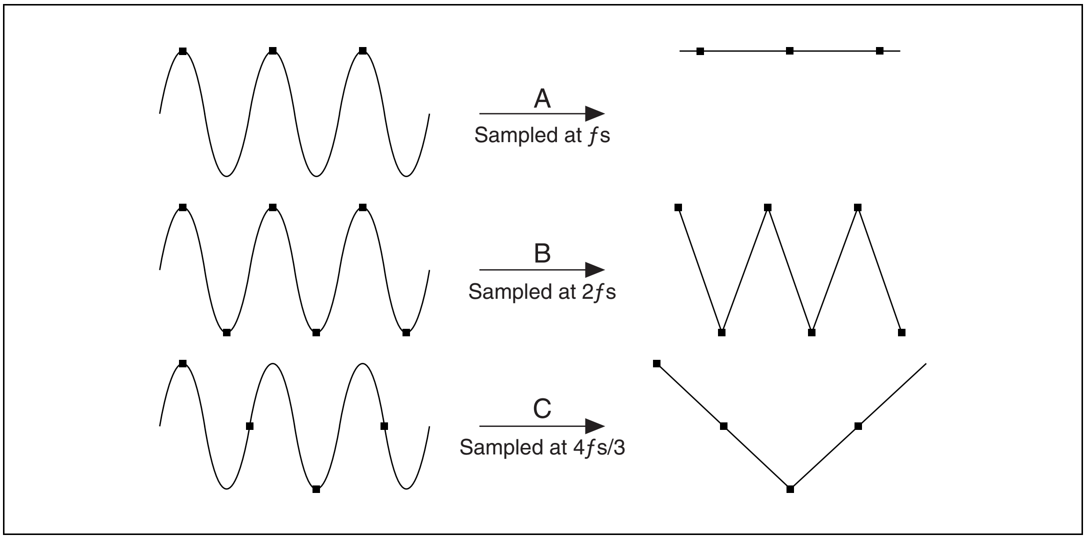


Figure 4-12. Effects of Various Sampling Rates

Example A samples the sine wave of frequency *f* at the same frequency *fs*. The acquired samples result in an alias at DC. However, if you increase the sampling rate to *2fs*, the digitized waveform has the correct frequency or same number of cycles as the original waveform but appears as a triangle waveform as shown in example B. By increasing the sampling rate to well above *fs*, you can more accurately reproduce the waveform. In example C, the sampling rate is 4*fs*/3. Because in this case the Nyquist frequency is below *fs*, (4*fs* /3 × 1)/2 = 2*fs* /3, this sampling rate reproduces an alias waveform of incorrect frequency and shape.

Digital I/O

An analog signal continuously varies with respect to time. A digital, or binary, signal has only two possible discrete levels—high level (ON) or low level (OFF). Figure 4-13 illustrates the main signal types.

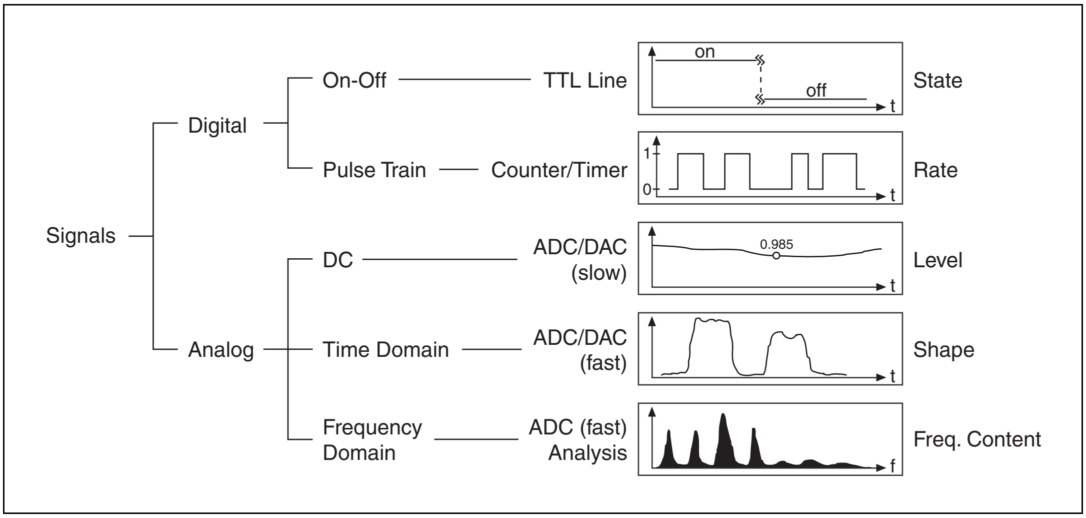


Figure 4-13. Signal Types

One example of a digital signal is a transistor-to-transistor logic (TTL) signal. A TTL signal has the following characteristics, as shown in Figure 4-14:

- 0 V–0.8 V = logic low
- 2 V–5 V = logic high
- Maximum Rise/Fall Time = 50 ns

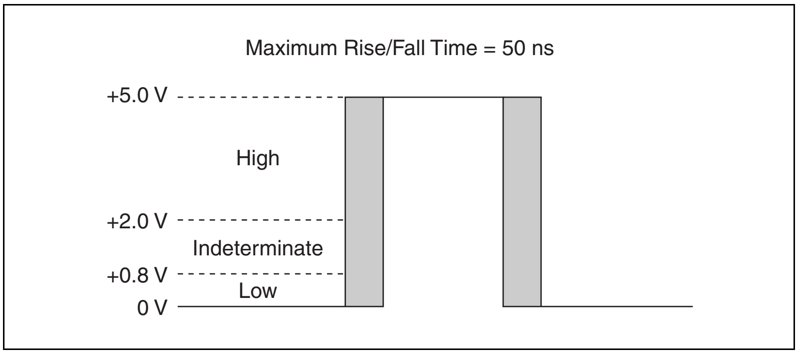


Figure 4-14. TTL Signal

Digital Lines and Ports

Digital lines and ports are important parts of a digital input/output system.

A line is an individual signal and refers to a physical terminal. The data that a line carries are called bits, which are binary values that are 1 or 0. The terms line and bit are interchangeable.

A port is a collection of digital lines. Usually the lines are grouped into an 8-bit port, in other words, a port with eight lines. Most E Series devices have one 8-bit port. The port width refers to the number of lines in a port. For example, if one port has eight lines, the port width is eight.

Handshaking

Use handshaking to communicate with an external device by using an exchange of signals to request and acknowledge each data transfer.

For example, using handshaking to acquire an image from a scanner includes the following steps.

1. The scanner sends a pulse to the measurement device after it scans the image and is ready to transfer the data.
2. The measurement device reads an 8-, 16-, or 32-bit digital sample.
3. The measurement device then sends a pulse to the scanner to let the scanner know it has read the digital sample.
4. The scanner sends out another pulse when it is ready to send another digital sample.
5. After the measurement device receives this digital pulse, it reads the sample.

This process repeats until all the samples are transferred.

Note Not all devices support handshaking. Refer to the device documentation for information about handshaking support. For E Series devices, only those with more than eight digital lines—those that have an additional onboard 8255 chip—support handshaking.

Triggering

A trigger is a signal that causes an action, such as starting the acquisition of data. Use a trigger if you need to set a measurement to start at a certain time. For instance, imagine that you want to test the response of a circuit board to a pulse input. You can use that pulse input as a trigger to tell the measurement device to start acquiring samples. If you do not use this trigger, you have to start acquiring data before you apply the test pulse.

When you configure a trigger, you must make two main decisions—what action you want the trigger to cause and how to produce the trigger.

If you want the trigger to begin the measurement, use a start trigger. If you want to acquire data before the trigger occurs, use a reference trigger, also known as a stop trigger, to capture samples before and after a trigger point, which becomes the reference position in the samples.

In addition to specifying the action you want a trigger to cause, you need to determine the source of the trigger. If you need to trigger off an analog signal, use an analog edge trigger or an analog window trigger. If the trigger signal is digital, you can use a digital edge trigger with a PFI pin as the source.

Analog Edge Triggering

An analog edge trigger occurs when an analog signal meets a condition you specify, such as the signal level or the rising or falling edge of the slope. When the measurement device identifies the trigger condition, it performs the action you associated with the trigger, such as starting the measurement or marking which sample was acquired when the trigger occurred.

In Figure 4-15, the trigger captures data for a rising edge signal when the signal reaches 3.2.

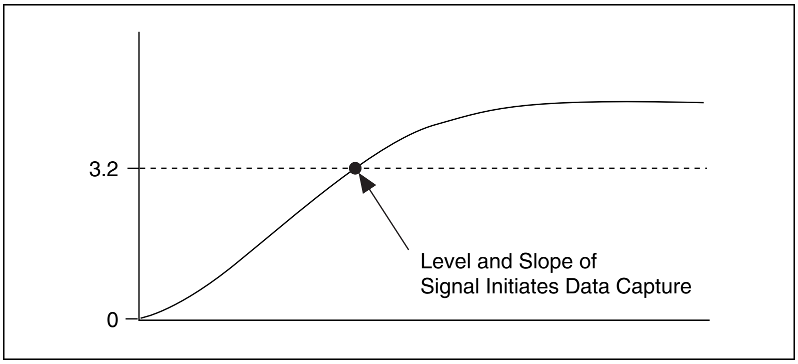


Figure 4-15. Analog Edge Triggering Example

Analog Window Triggering

An analog window trigger occurs when an analog signal passes into (enters) or passes out of (leaves) a window two voltage levels define. Specify the voltage levels by setting the window top value and the window bottom value.

In Figure 4-16, the trigger acquires data when the signal enters the window.

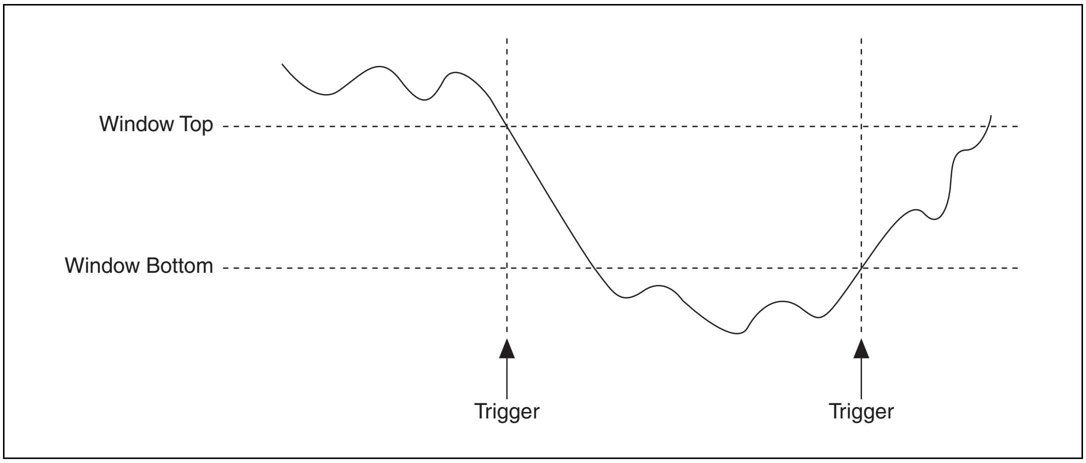


Figure 4-16. Enter Analog Window Triggering Example

In Figure 4-17, the trigger acquires data when the signal leaves the window.

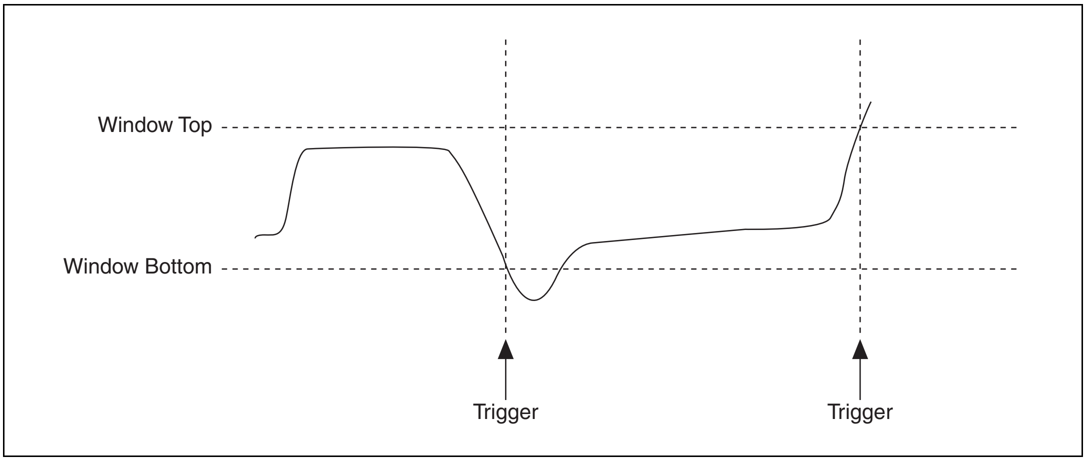


Figure 4-17. Leave Analog Window Triggering Example

Digital Edge Triggering

A digital edge trigger is usually a TTL signal that has two discrete levels: a high level and a low level. A digital signal creates a falling edge when it moves from a high level to a low level. The signal creates a rising edge when it moves from a low level to a high level. You can produce start or reference triggers based on the rising edge or falling edge of a digital signal. In Figure 4-18, the acquisition begins after the falling edge of the digital trigger signal. You usually connect digital trigger signals to PFI pins on a National Instruments measurement device.

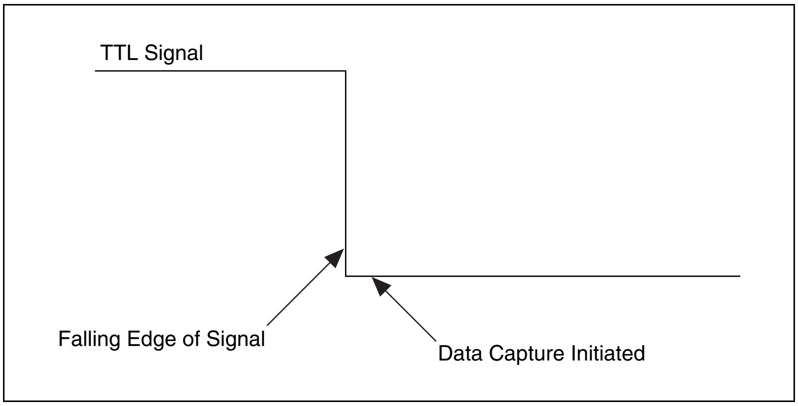


Figure 4-18. Digital Trigger

Signal Analysis

Signal analysis is the process of transforming an acquired signal to extract information about the signal, filter noise from the signal, and present the signal in a more understandable form than the raw signal.

Filtering and windowing are two signal analysis techniques. Refer to the *LabVIEW Analysis Concepts* manual or more information about signal analysis.

Filtering

Filtering is one of the most commonly used signal processing techniques. Signal conditioning systems can filter unwanted signals or noise from the signal you are measuring. Use a noise filter on low-rate, or slowly changing, signals, such as temperature, to eliminate higher frequency signals that can reduce signal accuracy. A common use of a filter is to eliminate the noise from a 50 or 60 Hz AC power line. A lowpass filter of 4 Hz removes the 50 or 60 Hz AC noise from signals sampled at low rates. A lowpass filter eliminates all signal frequency components above the cutoff frequency. Many signal conditioning modules have lowpass filters that have software-selectable cutoff frequencies from 10 Hz to 25 kHz.

Refer to Chapter 4, *Digital Filtering*, of the *LabVIEW Analysis Concepts* manual for more information about filtering.

Windowing

Use windowing, or smoothing windows, to minimize spectral leakage associated with truncated waveforms.

Spectral Leakage

Spectral leakage is a phenomenon whereby the measured spectral energy appears to leak from one frequency into other frequencies. It occurs when a sampled waveform does not contain an integral number of cycles over the time period during which it was sampled. The technique used to reduce spectral leakage is to multiply the time-domain waveform by a window function.

Discrete Fourier Transform (DFT) and Fast Fourier Transform (FFT) are mathematical techniques that resolve a given signal into the sum of sines and cosines. It is the basis for spectrum analysis. Using the DFT/FFT when you sample a noninteger number of cycles, such as 7.5 cycles, returns a spectrum in which it appears as if the energy at one frequency leaks into all the other frequencies because the FFT assumes that the data is a single period of a periodically repeating waveform. The artificial discontinuities appear as very high frequencies that were not present in the original signal. Because these frequencies are higher than the Nyquist frequency, they appear aliased between 0 and *fs*/2.

The type of window to use depends on the type of signal you acquire and on the application. Choosing the correct window requires some knowledge of the signal that you are analyzing. Table 4-2 lists common types of windows, the appropriate signal types, and example applications.

Table 4-2. Common Windows

Window	Signal Type and Description	Applications
Rectangular (no window)	Transient signals that are shorter than the length of the window; truncates a window to within a finite time interval	Order tracking, system analysis (frequency response measurements) with pseudorandom excitation, separation of two tones with frequencies very close to each other but with almost equal amplitudes
Triangle	Window that is the shape of a triangle	General-purpose applications
Hanning	Transient signals that are longer than the length of the window	General-purpose applications, system analysis (frequency response measurements) with random excitation
Hamming	Transient signals that are longer than the length of the window; a modified version of the Hanning window that is discontinuous at the edges	Often used in speech signal processing
Blackman	Transient signals; similar to Hanning and Hamming windows but adds one additional cosine term to reduce ripple	General-purpose applications
Flat Top	Has the best amplitude accuracy of all the windows but limits frequency selectivity	Accurate, single-tone amplitude measurements with no nearby frequency components

Note In many cases, you might not have sufficient knowledge of the signal, so you need to experiment with different windows to find the best one.

Device Calibration

Calibration consists of verifying the measurement accuracy of a device and adjusting for any measurement error. Verification consists of measuring the performance of the device and comparing these measurements to the factory specifications. During calibration, you supply and read voltage levels using external standards, and you adjust the device calibration constants. The device stores the new calibration constants in the EEPROM and loads the calibration constants from memory as needed to adjust for the error in the measurements taken by the device. Two kinds of calibration exist: external and internal, or self-calibration. For more information about calibrating a device, refer to the National Instruments Web site at `ni.com/calibration`.

External Calibration

External calibration, which a metrology lab typically performs, requires using a high-precision voltage source to verify and adjust calibration constants. This procedure replaces all calibration constants in the EEPROM and is equivalent to a factory calibration. Because the external calibration procedure changes all EEPROM constants, it invalidates the original National Institute of Standards and Technology (NIST) traceability certificate. If an external calibration is performed with a NIST-certified voltage source, a new NIST traceability certificate can be issued.

Internal Calibration

Internal calibration, or self-calibration, uses a software command and requires no external connections. Self-calibration adjusts a device for use in an environment where external variables, such as temperature, might differ from those in the environment in which the device was externally calibrated.

5

Creating a Typical Measurement Application

This chapter introduces you to common LabVIEW features you can use to develop measurement applications that acquire, analyze, and present measurement data.

I/O Controls

Use the I/O controls located on the **I/O** palette to specify the instrument or device resource you want to communicate with. The control you select depends on the instrument or device. Wire the I/O terminal on the block diagram to the channel or string terminal of a Traditional NI-DAQ, NI-DAQmx, IVI, VISA, FieldPoint, or Motion VI. You must install the necessary drivers and attach the necessary devices to the computer before using I/O name controls.

Note All I/O name controls and constants are available on all platforms. However, if you try to run a VI with a platform-specific I/O control on a platform that does not support that device, you will receive an error.

Traditional DAQ Channel Control

If you use Traditional NI-DAQ to control a DAQ device, use the Traditional DAQ Channel control to access the channels you configure using MAX or another configuration utility.

Any channels you configure appear as options in the Traditional DAQ Channel control pull-down menu. Right-click the control and select **I/O Name Filtering** from the shortcut menu to filter channels based on configuration,

(Windows) To create a new channel using MAX, right-click the control and select **Create New Channel** from the shortcut menu to launch MAX.

DAQmx Name Controls

If you use NI-DAQmx to control a DAQ device, use the controls on the **DAQmx Name Controls** palette to access the tasks, scales, devices, global channels, and switches you configure using MAX and the DAQ Assistant. Right-click the control and select **I/O Name Filtering** from the shortcut menu to filter the options based on configuration.

Refer to the *Physical and Virtual Channels* section of this chapter for more information about using NI-DAQmx tasks.

IVI Logical Name Control

Use the IVI Logical Name control with IVI instrument drivers to access the logical names you configure using MAX. Logical names appear in the IVI Logical Name control pull-down menu and are references to instruments that use IVI instrument drivers. The IVI Logical Name Control also displays VISA resource names for using specific IVI drivers without using MAX.

VISA Resource Name Control

Use the VISA resource name control to access the VISA aliases you configure using MAX. VISA aliases and VISA resource names appear in the VISA resource name control pull-down menu.

FieldPoint I/O Point Control

Use the FieldPoint I/O Point control to access the FieldPoint items you create and configure using MAX. Any items you configure in MAX appear as options in the FieldPoint I/O Point control pull-down menu.

Motion Resource Name Control

Use the Motion Resource Name control to access a motion resource you configure using MAX. Right-click the control and select **Allow Undefined Names** from the shortcut menu to use names without using MAX.

Polymorphic VIs

Many of the Traditional NI-DAQ and NI-DAQmx VIs are polymorphic and can accept or return data of various types, such as scalar values, arrays, or waveforms. You use other polymorphic NI-DAQmx VIs to configure various triggers and methods of sample timing, and to create virtual channels. By default, NI-DAQmx VIs appear with the polymorphic VI selector.

Refer to the *Polymorphic VIs and Functions* section of Chapter 5, *Building the Block Diagram*, of the *LabVIEW User Manual* for more information about polymorphic VIs.

Properties

You can write most applications using only the VIs of the NI-DAQmx, NI-VISA, and IVI Instrument Driver APIs. You also can use properties with these APIs to extend the functionality to include less commonly used features. For example, you can use the VISA Configure Serial Port VI to set several commonly used serial port settings in a VISA Session, including the baud rate. However, if you want to change only the baud rate, you can use a Property Node.

Use the Property Node on the **DAQmx** palette to configure various low-level settings for NI-DAQmx. Use the Property Node on the **VISA Advanced** palette for any VISA property. Use the Property Nodes on the **Modular Instrument** palette and the **IVI Instrument Drivers** palette for these APIs, respectively.

Creating a Typical DAQ Application

Use the VIs on the **NI Measurements** palette to develop DAQ applications. Follow the basic programmatic steps outlined in Figure 5-1 when you create a DAQ application.

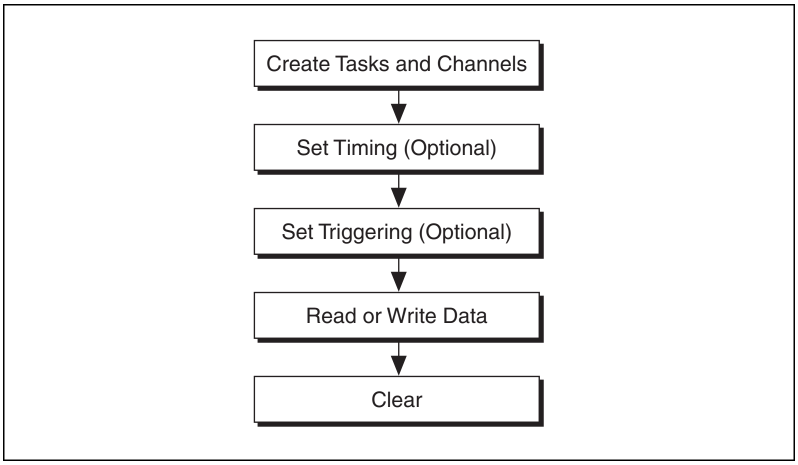


Figure 5-1. Basic Steps in Creating a DAQ Application

Notice that timing and triggering are optional. Include the timing step if you want to specify hardware timing instead of software timing. If you are using NI-DAQmx, you can use the DAQ Assistant to set timing parameters for a task.

Use triggering if you want the device to acquire samples only when certain conditions are met. For example, you might want to acquire samples if the input signal goes higher than 4 V. If you are using NI-DAQmx, you can use the DAQ Assistant to configure triggering for a task.

Many NI-DAQmx applications also can include steps to start, stop, and clear the task. For instance, for applications that use a counter/timer to count edges or to measure period, use the Start VI to arm the counter.

In NI-DAQmx, LabVIEW clears the task automatically when the VI hierarchy that created the task finishes executing.

Traditional NI-DAQ and NI-DAQmx include VIs for timing, triggering, reading, and writing samples. You can use the NI-DAQmx properties to extend the functionality of the NI-DAQmx VIs. Refer to the *Data Acquisition VIs for Traditional NI-DAQ* help for more information about using the Traditional NI-DAQ VIs. Refer to the *DAQmx Data Acquisition VIs* help for more information about using the NI-DAQmx VIs and properties.

Physical and Virtual Channels

A physical channel is a terminal or pin at which you can measure or generate an analog or digital signal. Every physical channel on a device that supports NI-DAQmx has a unique name.

A virtual channel is a collection of property settings that can include a name, a physical channel, input terminal connections, the type of measurement or generation, and scaling information. In Traditional NI-DAQ and earlier versions, configuring virtual channels is an optional way to record which channels are being used for different measurements, but virtual channels are integral to every NI-DAQmx measurement.

Tasks

A task in NI-DAQmx is a collection of one or more virtual channels with timing, triggering, and other properties. A task represents a measurement or a generation you want to perform. You can set up and save all of the configuration information in a task and use the task in an application.

In NI-DAQmx, you can configure virtual channels as part of a task or separate from a task.

Complete the following steps to perform a measurement or a generation with a task.

1. Create a task and channels.
2. (Optional) Configure the channel, timing, and triggering properties.
3. Read or write samples.
4. Clear the task.

Repeat steps 2 and 3, if it is appropriate for the application. For instance, after reading or writing samples, you can reconfigure the channel, timing, or triggering properties and then read or write additional samples based on this new configuration.

Waveform Control and Digital Waveform Control

Use the Waveform control, the Digital Waveform control, the Waveform Graph, and the Digital Waveform Graph to represent the waveforms and digital waveforms you acquire or generate. LabVIEW represents an analog waveform, such as a sine wave or a square wave, with the waveform data type by default. A 1D array of waveform data type represents multiple waveforms. LabVIEW represents a digital waveform with the digital waveform data type by default.

The waveform and digital waveform controls consist of components that include a start time, a delta t, the waveform data, and attributes. Use the Waveform VIs and functions to access and manipulate individual components.

Start Time (t0)

The start time (t0) is a timestamp associated with the first measurement point in the waveform. Use the start time to synchronize plots on a multi-plot waveform graph or digital waveform graph and to determine delays between waveforms.

Delta t (dt)

Delta t (dt) is the time interval between any two points in the signal.

Waveform Data and Digital Waveform Data (Y)

The waveform data and the digital waveform data are the values that represent the waveform.

An array of any numeric data type can represent analog waveform data. Generally, the number of data values in the array corresponds directly to the number of scans from a DAQ device.

The digital data type represents a digital waveform and displays the digital data in a table.

Attributes

Attributes include information about the signal, such as the name of the signal and the device acquiring the signal. NI-DAQ automatically sets some attributes for you. Use the Set Waveform Attribute function to set attributes, and use the Get Waveform Attribute function to read attributes.

Displaying Waveforms

To represent waveform data on the front panel, use the Waveform control or the Waveform Graph. To represent digital waveform data, use the Digital Waveform control or the Digital Waveform Graph.

Use the Waveform control and the Digital Waveform control to manipulate the t0, dt, and Y components of the waveform or to display those components as an indicator.

When you wire a waveform to a graph, the t0 component is the initial value on the x-axis. The number of scans acquired and the dt component determine the subsequent values on the x-axis. The data elements in the Y component comprise the points on the plot of the graph.

If you want to let a user control a certain component, such as the dt component, create a front panel control and wire it to the appropriate component in the Build Waveform function.

The VI in Figure 5-2 continuously acquires 10,000 scans from a DAQ device at a sample rate of 1,000 scans/second, which began at 7:00 p.m. The graph plots the waveform data (Y) on the graph. The start time (t0) is 7:00:00 p.m. and is the first point on the x-axis. The delta t (dt) of the waveform is 1.00 ms (1,000 scans/second = 1 ms/scan), so the 10,000 scans are distributed over 10 seconds with the last data value plotted at 7:00:10 p.m.

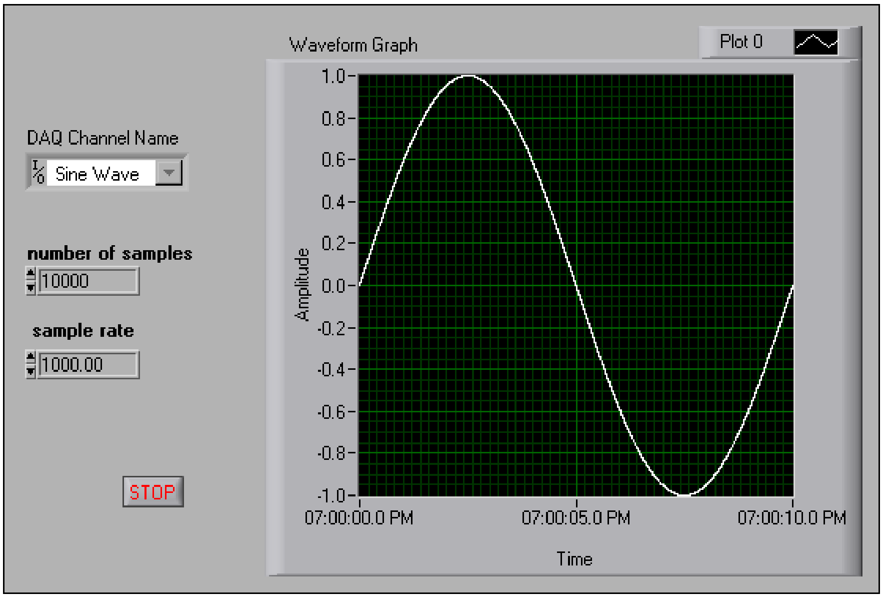


Figure 5-2. Waveform Graph

Using the Waveform Control

Several VIs accept, operate on, and/or return waveforms. In addition, you can wire the waveform data type directly to many controls, including the graph, chart, numeric controls, and numeric array controls.

The block diagram in Figure 5-3 acquires a waveform from a channel on a DAQ device, filters the signal, and plots the resulting waveform on a graph.

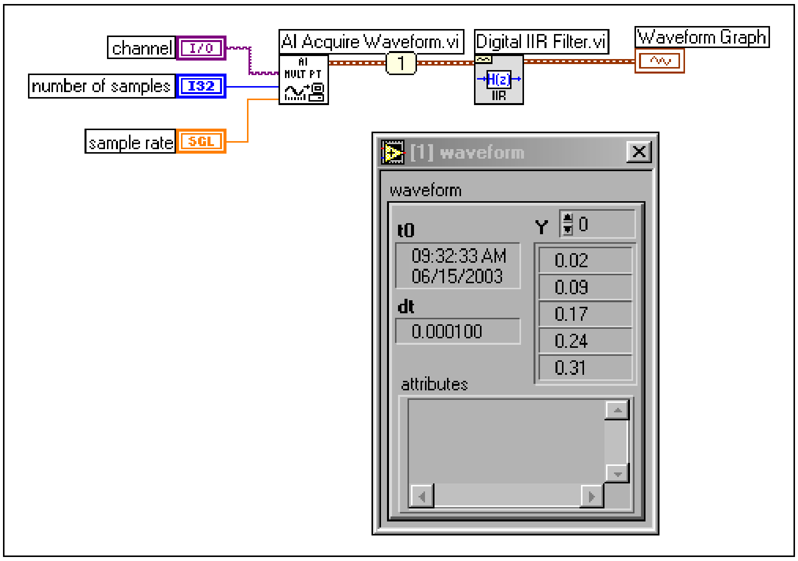


Figure 5-3. Using the Waveform Data Type

The AI Acquire Waveform VI acquires a specified number of samples at a specified sample rate at a particular time from a single input channel and returns a waveform. The probe displays the components of the waveform data type, which include the time the acquisition began (t0), the time between successive data points (dt), and the data of a waveform acquired with each scan (Y). The Digital IIR Filter VI accepts the array of waveforms and filters the data (Y) of each waveform. The waveform graph plots and displays the waveform.

You also can use the waveform data type with single-point acquisitions, as shown in Figure 5-4.

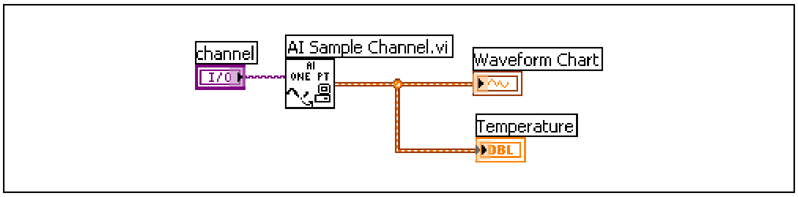


Figure 5-4. Waveform Data Type and Single-Point Acquisitions Example

The AI Sample Channel VI acquires a single sample from a channel and returns a single-point waveform. The waveform contains the value read from the channel and the time the channel was read. The chart and the temperature indicator accept the waveform and display its data.

You also can use the waveform data type with analog output, as shown in Figure 5-5. The Sine Waveform VI generates a sine waveform, and the AO Generate Waveform VI sends the waveform to the device.

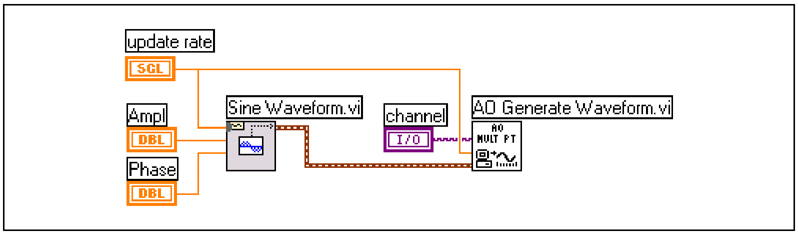


Figure 5-5. Using the Waveform Data Type with Analog Output

Extracting Waveform Components

Use the Get Waveform Components function to extract and manipulate the components of a waveform you generate. The block diagram in Figure 5-6 uses the Get Waveform Components function to extract the waveform data. The Negate function negates the waveform data and plots the results to a graph.

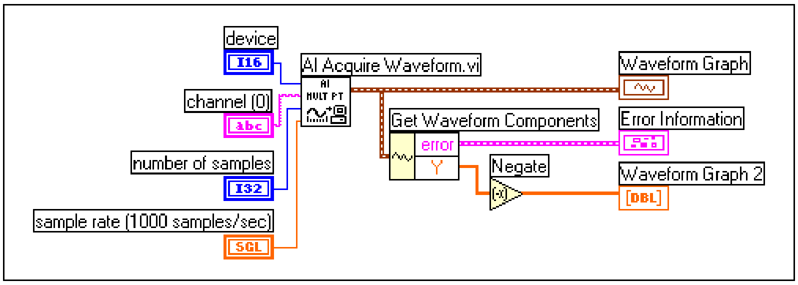


Figure 5-6. Extracting Waveform Components

Using the Digital Waveform Control

Use the VIs and functions on the **Digital Waveform** palette to manipulate digital data by extracting and editing the components of the digital signal. Use the NI-DAQmx VIs on the **Digital I/O** palette to acquire and send a digital signal. The **Digital Waveform** palette also includes VIs that convert analog data to digital signals, search a digital signal for a pattern, append a digital signal(s) to another digital signal, and perform other digital tasks.

Creating a Typical VISA Application

Use the VIs and functions on the **VISA** palette to build VIs that control instruments. Refer to the *VISA in LabVIEW* section of Chapter 16, *Using LabVIEW to Control Instruments*, for more information about creating VISA VIs.

Creating a Typical FieldPoint Application

Use the VIs on the **FieldPoint** palette to develop FieldPoint distributed I/O applications. Most FieldPoint applications require only the FP Read VI and the FP Write VI. The FP Read VI returns data from the FieldPoint I/O channel or group of channels the FieldPoint IO Point function represents. The FP Write VI sends data to the FieldPoint I/O channel or group of channels the FieldPoint IO Point function represents.

Channels versus Items

FieldPoint modules consist of physical I/O points called channels. Items represent the channels or groups of channels. You can create items in MAX and use the FieldPoint I/O Point control to access the items in LabVIEW.

Using the FieldPoint I/O Point Control

Use the FieldPoint I/O Point control to communicate with the FieldPoint items you create in MAX. Place the FieldPoint I/O Point control on the front panel, right-click the control, and select the items you want to read from or write to from the shortcut menu. If you do not see the FieldPoint items you want, configure the items in MAX.

6 Measuring DC Voltage

This chapter describes how to measure DC voltage using DAQ devices and instruments.

Overview of DC Measurements

There are two types of voltage: direct current (DC) and alternating current (AC). DC signals are analog signals that slowly vary with time. Common DC signals include voltage, temperature, pressure, and strain. AC signals are alternating analog signals that continuously increase, decrease, and reverse polarity on a repetitive basis.

Refer to Chapter 7, *Measuring AC Voltage*, for more information about AC measurements.

DC applications dominated the early days of high-voltage electricity. The constant nature of DC made it easy to measure voltage, current, and power. The power formulas for DC are $P = I^2 \times R$ and $P = V^2/R$, where P is power (watts), I is current (amps), R is resistance (ohms), and V is voltage (volts DC).

Using NI-DAQ VIs to Measure DC Voltage

With DC signals, you are most interested in how accurately you can measure the amplitude of a signal at a given point in time. Use signal conditioning to improve the accuracy of most measurements. Refer to the *Signal Conditioning* section of Chapter 4, *Measurement Fundamentals*, for more information about signal conditioning.

Figure 6-1 shows a typical wiring diagram for an anemometer with an output range of 0 to 10 V, which corresponds to wind speed from 0 to 200 mph. Use the following equation to scale the data:

$$\text{anemometer reading}(\mathrm{V}) \times 20\left(\frac{\mathrm{mph}}{\mathrm{V}}\right) = \text{ wind speed (mph)}$$

Notice that the wiring diagram in Figure 6-1 uses a resistor, R, because an anemometer is usually not a grounded signal source. If the anemometer transducer were already grounded, using R would cause a ground loop and would result in erroneous readings.

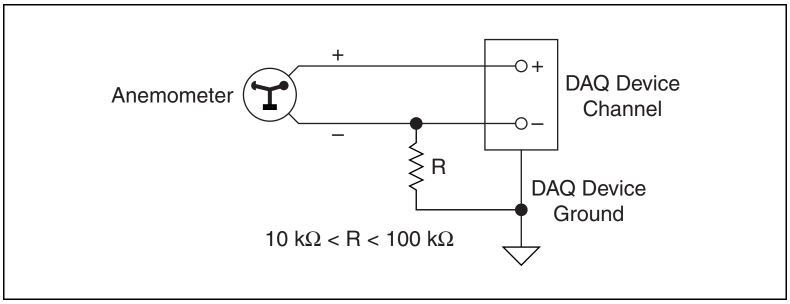


Figure 6-1. Anemometer Wiring

Traditional NI-DAQ Method

The block diagram in Figure 6-2 uses Traditional NI-DAQ to measure wind speed. **device** is the number assigned to the plug-in DAQ device during configuration. **Channel** is the analog input channel the anemometer is wired to. The **high limit** and **low limit** values show the expected voltage range, which determines the amount of gain the DAQ device applies. The AI Sample Channel VI acquires a single value, in this case, raw voltage. The scaling value of 20 mph/volt wired to the Multiply function scales the input voltage range of 0 V to 10 V to the wind speed range of 0 mph to 200 mph.

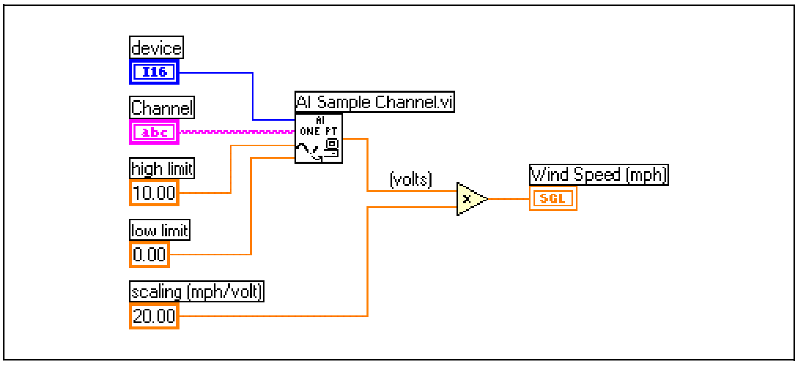


Figure 6-2. Measuring Voltage and Scaling to Wind Speed

Use DAQ Named Channels to simplify this block diagram as shown in Figure 6-3. The DAQ Named Channel in Figure 6-3 includes information about the device, channel, gain, and the scaling equation. The AI Sample Channel VI acquires a single value, but in this case, it returns the wind speed.

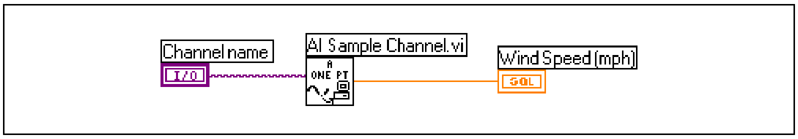


Figure 6-3. Measuring Wind Speed Using DAQ Named Channels

NI-DAQmx Method

The block diagram in Figure 6-4 uses NI-DAQmx VIs to measure wind speed. The DAQmx Create Virtual Channel VI uses the **Physical Channel** to create an Analog Input Voltage virtual channel. The voltage range is 0 to 10 V. The DAQmx Read VI reads one sample from a single channel. The scaling value of 20 mph/volt wired to the Multiply function scales the input voltage range of 0 V to 10 V to the wind speed range of 0 to 200 mph.

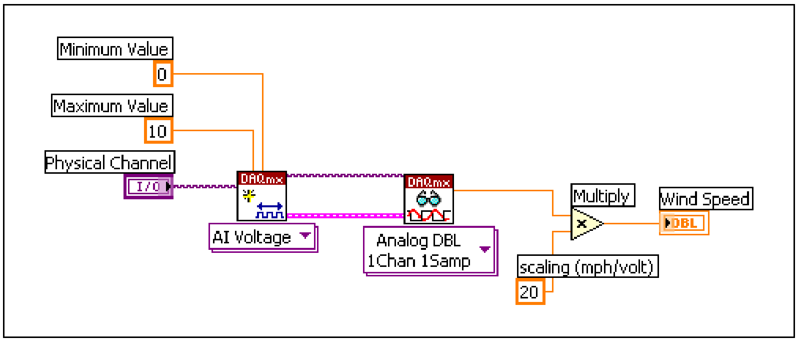


Figure 6-4. Acquiring Single Voltage Readings Using DAQmx VIs

Averaging a Scan

Averaging can yield a more useful reading if a signal is rapidly changing or if noise exists on the line.

Figure 6-5 shows what an actual wind speed might look like over time. Due to gusting winds, the speed values look noisy. Notice that the wind speed reading of 29 mph is a peak speed that might give the impression that the wind is holding at 29 mph. A better representation might be to take the average speed over a short period of time.

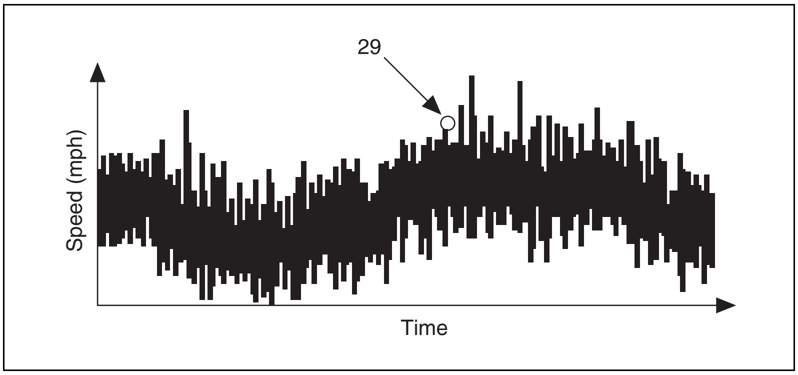


Figure 6-5. Wind Speed

Figure 6-6 shows a DAQ system for measuring wind speed with the addition of software averaging.

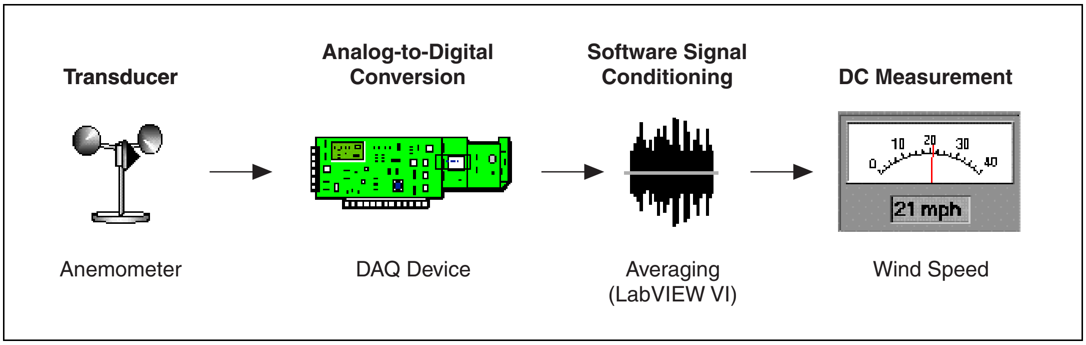


Figure 6-6. DAQ System for Measuring Wind Speed with Averaging

Traditional NI-DAQ Method

The block diagram in Figure 6-7 uses Traditional NI-DAQ and DAQ Named Channels to measure an average wind speed. Notice that this block diagram uses the AI Acquire Waveform VI to acquire a waveform instead of a single value. The **number of samples** and **sample rate** inputs define the waveform of data acquired. For example, if you set **number of samples** to 1,000 and **sample rate** to 500 (samples/sec), the VI takes two seconds to acquire the 1,000 points. The Mean VI returns the average wind speed for two seconds of time.

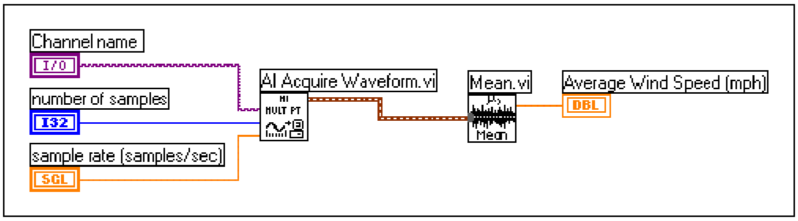


Figure 6-7. Averaging Wind Speed Using DAQ Named Channels

One common reason for averaging is to eliminate 50 or 60 Hz powerline noise. The oscillating magnetic field around powerlines can introduce noise voltages on unshielded transducer wiring. Because powerline noise is sinusoidal, or shaped like a sine wave, the average over one period is zero. If you use a scan rate that is an integer multiple of the noise and average data for an integer multiple of periods, you can eliminate the line noise. One example that works for both 50 and 60 Hz is to scan at 300 scans per second and average 30 points. Notice that 300 is an integer multiple of both 50 and 60. One period of the 50 Hz noise is 300/50 = 6 points. One period of the 60 Hz noise is 300/60 = 5 points. Averaging 30 points is an integer multiple of both periods, so you can ensure that you average whole periods.

NI-DAQmx Method

The block diagram in Figure 6-8 uses NI-DAQmx VIs to average a signal. This block diagram uses the Analog Wfm 1Chan NSamp instance of the DAQmx Read VI to acquire multiple values from a single channel. The DAQmx Read VI reads 1,000 samples from the virtual channel that the DAQmx Create Virtual Channel VI returns. The Mean VI averages the 1,000 samples from the DAQmx Read VI and returns the average wind speed.

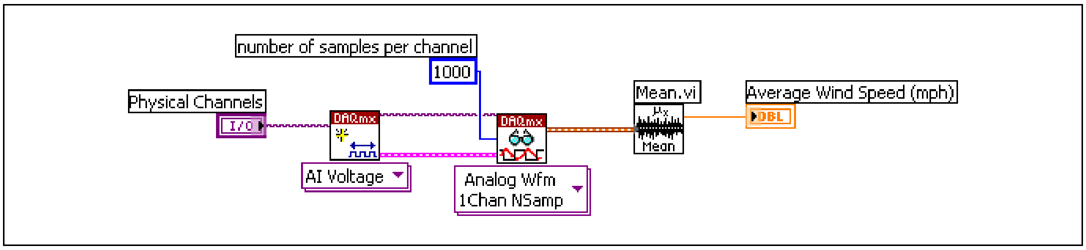


Figure 6-8. Averaging Multiple Samples Using NI-DAQmx VIs

Measuring DC Voltage with Instruments

Figure 6-9 shows a measurement system that uses a stand-alone instrument for a DC voltage measurement. The stand-alone instrument also could be an instrument device that plugs directly into a PC.

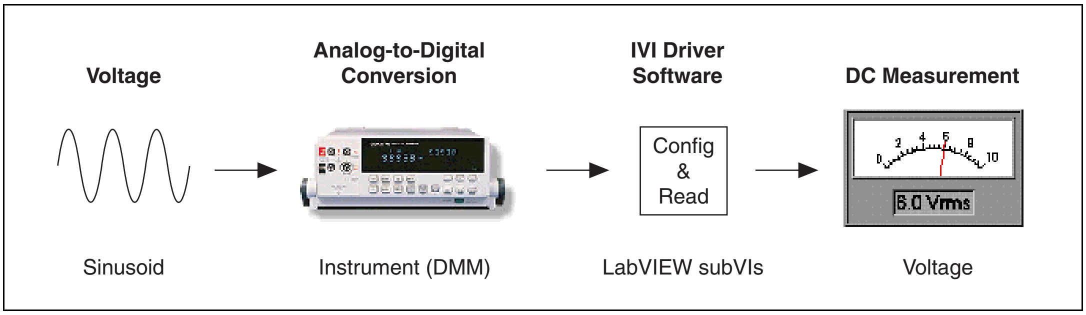


Figure 6-9. Instrument Control System for DC Voltage

The block diagram in Figure 6-10 uses the IVI class driver VIs to measure DC voltage. The IviDmm Initialize VI uses a logical name to create a session and initialize the instrument. The IviDmm Configure Measurement VI configures the measurement for DC volts. The IviDmm Read VI takes the measurement, and the IviDmm Close VI closes the session.

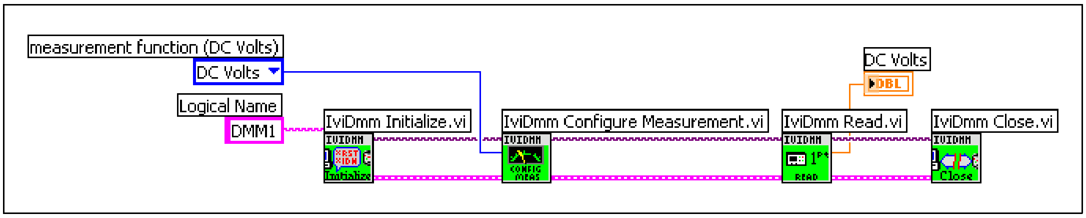


Figure 6-10. Measuring DC Voltage Using IVI Class Driver VIs

A common, but inefficient, way to read and average multiple points is to place a For Loop around the IviDmm Read VI, but many DMMs can read and average multiple points inside the instrument, which is more efficient.

7

Measuring AC Voltage

This chapter describes how to measure AC voltage using DAQ devices, FieldPoint modules, and instruments.

Overview of AC Measurements

Most power lines today deliver AC for home, lab, and industrial applications. Voltage, current, and power are not constant values because AC signals alternate. However, it is useful to measure voltage, current, and power such that a load connected to a 120 volts AC (VAC) source develops the same amount of power as that same load connected to a 120 volts DC (VDC) source. For this reason, V_{rms} (root mean square) was developed. With RMS, the power formula for DC also works for AC. For sinusoidal waveforms, V_{rms} = Vpeak/square root of 2. Because voltmeters read V_{rms}, the 120 VAC of a typical U.S. wall outlet actually has a peak value of about 170 V.

Refer to Chapter 6, *Measuring DC Voltage*, for more information about DC measurements.

Measuring AC Voltage with Instruments

Figure 7-1 shows a DAQ system for measuring V_{rms}.

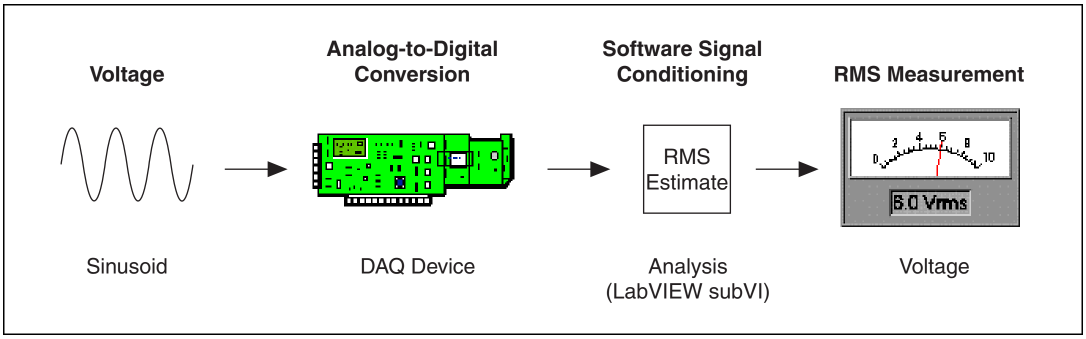


Figure 7-1. DAQ System for V_{rms}

Traditional NI-DAQ Method

The block diagram in Figure 7-2 uses a DAQ Named Channel to measure V_{rms}.

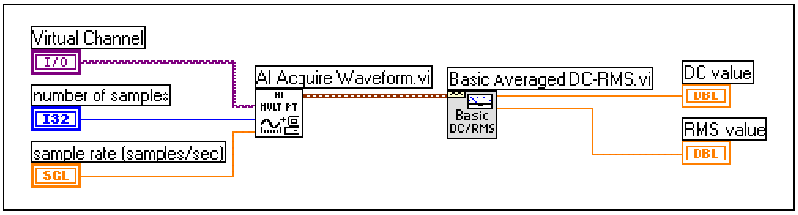


Figure 7-2. V_{rms} Using DAQ Named Channels

The AI Acquire Waveform VI acquires a waveform. The **number of samples** and **sample rate** inputs define the waveform. The Basic Averaged DC-RMS VI estimates the RMS and DC components. For a sinusoidal waveform centered at about zero, the Basic Averaged DC-RMS VI returns the **DC value** and V_{rms}. For a sinusoidal waveform offset from zero, the **DC value** indicates the DC shift, and the **RMS value** indicates V_{rms} as if the waveform were centered at about zero.

According to the Nyquist Theorem, you must sample at a rate greater than twice the maximum frequency component in the signal you are acquiring to accurately represent the signal. Refer to the *Aliasing* section of Chapter 4, *Measurement Fundamentals*, for more information about the Nyquist Theorem.

However, V_{rms} relates to the shape of the waveform, not the frequency of the data. To accurately acquire a waveform shape, you typically must acquire at five to 10 times the frequency of the waveform.

NI-DAQmx Method

The block diagram in Figure 7-3 uses NI-DAQmx VIs to acquire an AC voltage reading. The DAQmx Create Virtual Channel VI creates a virtual channel to acquire a voltage signal. The DAQmx Timing VI is set to Sample Clock with a finite sample mode. Samples per Channel and Rate determine how many samples per channel to acquire and at what rate. Because this example acquires 20,000 samples at a rate of 20,000 samples per second, the acquisition takes one second and terminates. The DAQmx Read VI measures the 20,000 voltage samples and returns this waveform to the Basic Averaged DC-RMS VI, which estimates the DC and RMS waveform values.

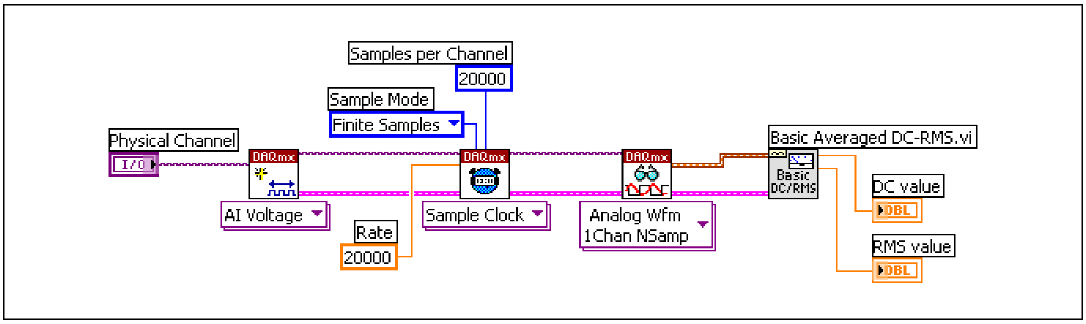


Figure 7-3. Measuring AC Voltage using NI-DAQmx VIs

Measuring Maximum, Minimum, and Peak-to-Peak Voltage with Instruments

Figure 7-4 shows a DAQ system with a signal that changes over time.

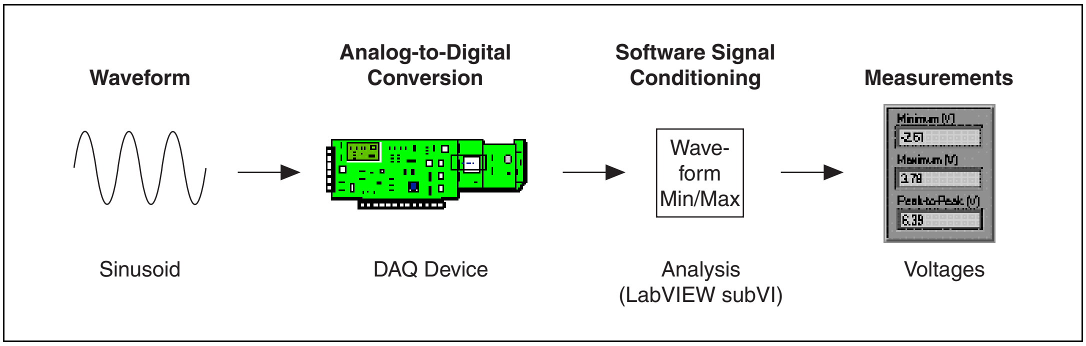


Figure 7-4. DAQ System for Minimum, Maximum, Peak-to-Peak

For this measurement, the signal typically might be repetitive, but reading the maximum, minimum, and peak-to-peak values does not require a repetitive signal. The peak-to-peak value is the maximum voltage swing (maximum – minimum).

The block diagram in Figure 7-5 uses Traditional NI-DAQ VIs to measure maximum, minimum, and peak-to-peak voltage values.

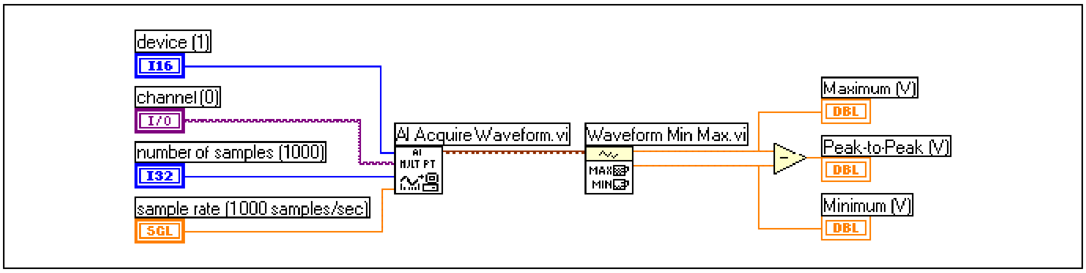


Figure 7-5. Using Traditional NI-DAQ to Measure Minimum, Maximum, and Peak-to-Peak Voltages

The AI Acquire Waveform VI scans data from one channel of a DAQ device. The Waveform Min Max VI returns the minimum and maximum values of the waveform. The difference of these values is the peak-to-peak voltage.

Using Instruments to Measure AC Voltage

Figure 7-6 shows a measurement system that uses a stand-alone instrument for an AC voltage measurement. The stand-alone instrument also could be an instrument that plugs directly into a PC.

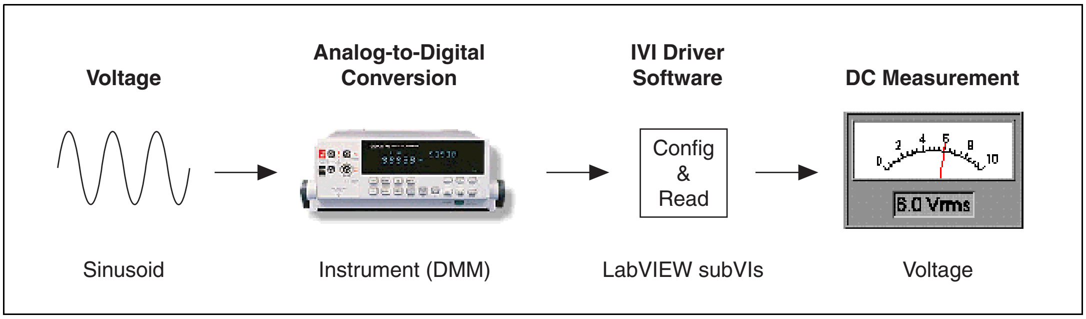


Figure 7-6. Instrument Control System for V_{rms}

The block diagram in Figure 7-7 uses the IVI class driver VIs to measure V_{rms}. The IviDmm Initialize VI uses a logical name to create a session and initialize the instrument. The IviDmm Configure Measurement VI configures the measurement for AC volts. The IviDmm Read VI takes the measurement, and the IviDmm Close VI closes the session.

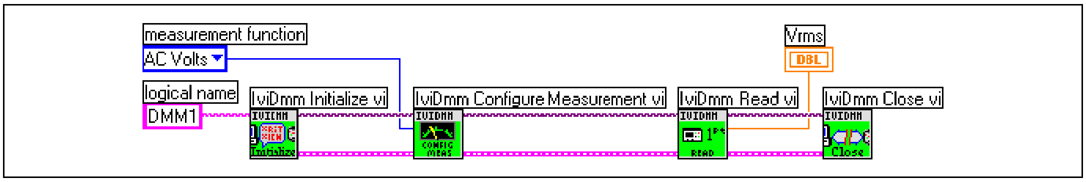


Figure 7-7. V_{rms} Using an Instrument

Using an Instrument to Measure Peak-to-Peak Voltage

Figure 7-8 shows a measurement system that uses a stand-alone instrument to measure peak-to-peak voltage. The stand-alone instrument also could be an instrument that plugs directly into a PC.

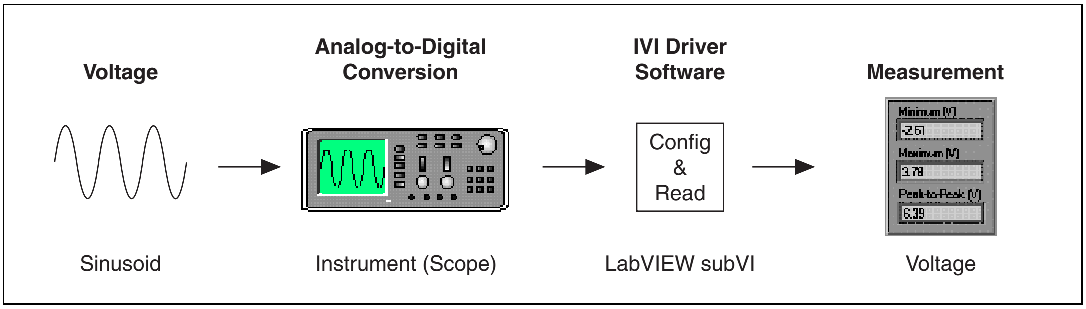


Figure 7-8. Instrument Control System for Peak-to-Peak Voltage

The block diagram in Figure 7-9 uses the IVI class driver VIs to measure peak-to-peak voltage. The IviScope Initialize VI uses a logical name to create a session and initialize the instrument. The IviScope Auto Setup [AS] VI configures many instrument settings. The IviScope Configure Channel VI sets the coupling to AC to remove the DC component of the signal. The IviScope Read Waveform Measurement [WM] VI takes the measurement, and the IviScope Close VI closes the session.

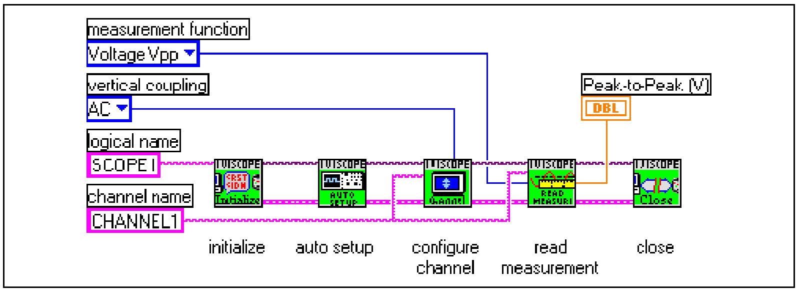


Figure 7-9. Measuring Peak-to-Peak Voltage Using IVI Class Driver VIs

Using FieldPoint VIs to Measure AC Voltage

Figure 7-10 shows a distributed FieldPoint system for measuring V_{rms}.

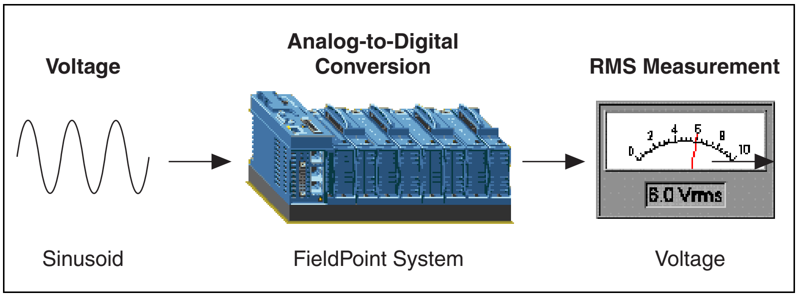


Figure 7-10. FieldPoint System for Measuring Voltage

The block diagram in Figure 7-11 uses a FieldPoint VI to measure V_{rms}. In this example, the FieldPoint I/O Point control represents the cFP-AI-102 FieldPoint module.

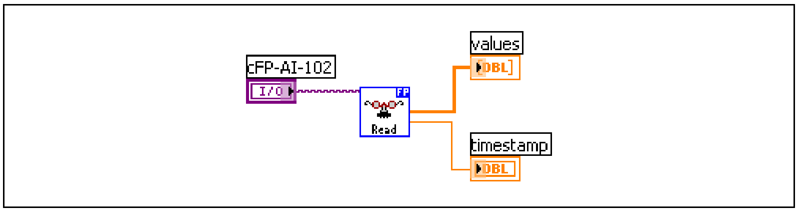


Figure 7-11. Measuring V_{rms} Using FieldPoint VIs

8 Measuring Temperature

This chapter describes how to measure temperature using DAQ devices, FieldPoint modules, and instruments.

Using NI-DAQ VIs to Measure Temperature

A popular way to measure temperature with a DAQ device is to use a thermocouple, as shown in Figure 8-1, because thermocouples are inexpensive, easy to use, and easy to obtain.

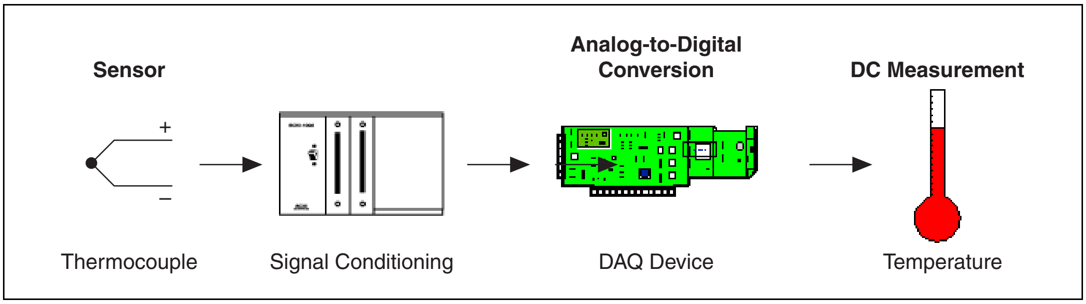


Figure 8-1. Simple Temperature System Using DAQ

A thermocouple forms when two dissimilar metals come in contact with each other and produce a temperature-related voltage. Refer to the National Instruments Web site at `ni.com/info` and enter the info code `ext4n9` for more information about using a thermocouple to measure temperature.

In the typical wiring diagram for a thermocouple shown in Figure 8-2, notice that the resistor, R, is used only if the thermocouple is not grounded at any other point. If, for example, the thermocouple tip were already grounded, using R would cause a ground loop and result in erroneous readings.

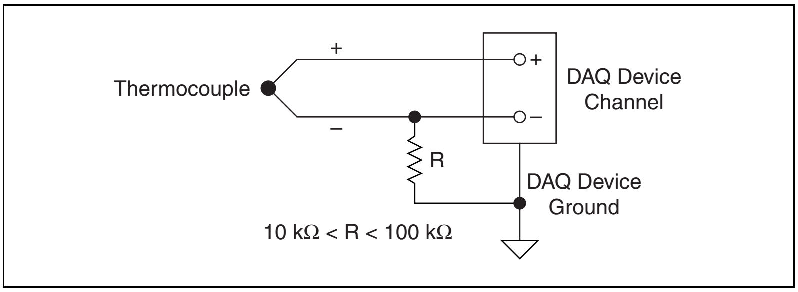


Figure 8-2. Thermocouple Wiring

Traditional NI-DAQ Method

The block diagram in Figure 8-3 uses DAQ Named Channels to measure temperature. In this example, the DAQ Named Channel handles all gain, linearization, and cold-junction compensation.

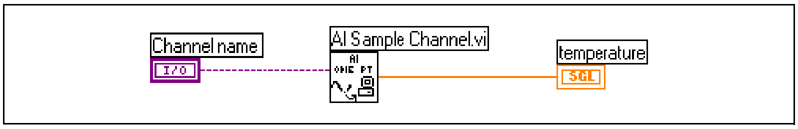


Figure 8-3. Measuring Temperature Using DAQ Named Channels

If you do not want to use DAQ Named Channels to measure temperature, you must write a VI that determines the gain needed for the temperature range, read the thermocouple voltage, read the cold-junction voltage, and convert all this information into a temperature. Refer to the Single Point Thermocouple Measurement VI in the `examples\DAQ\solution\transduc.llb` for an example of measuring temperature without using DAQ Named Channels.

NI-DAQmx Method

The block diagram in Figure 8-4 uses a DAQmx Task Name constant to measure temperature. In this example, a task configured in the DAQ Assistant and named My Temperature Task acquires the measurement. The task contains information like thermocouple type, cold-junction compensation (CJC) location and value, scaling information, and so on. The DAQmx Read VI measures and returns the temperature and graphs the data. By using a NI-DAQmx task, you can set up and edit configuration information without changing the block diagram.

Refer to the *Tasks* section of Chapter 5, *Creating a Typical Measurement Application*, for more information about tasks.

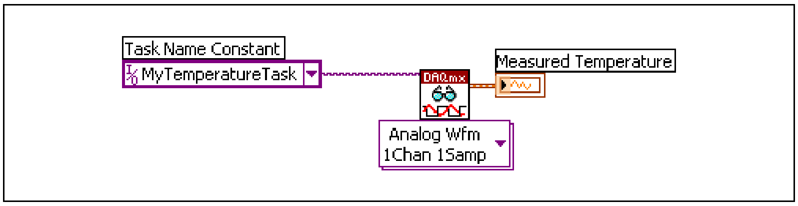


Figure 8-4. Measuring Temperature Using NI-DAQmx VIs

You also can use the DAQmx Create Channel VI to programmatically configure a temperature measurement.

Using FieldPoint VIs to Measure Temperature

Figure 8-5 shows a FieldPoint system for measuring a single temperature value.

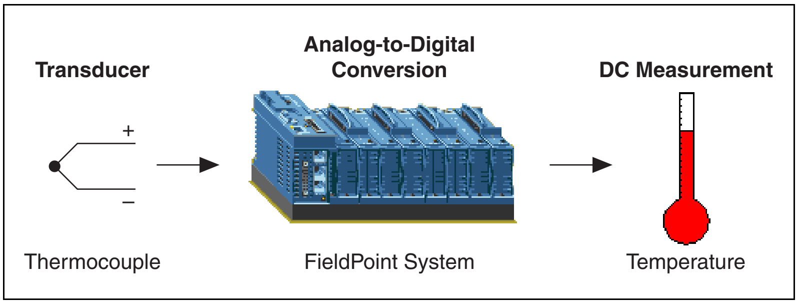


Figure 8-5. FieldPoint System for Measuring Temperature

The block diagram in Figure 8-6 uses a FieldPoint VI to measure temperature. In this example, the FieldPoint I/O Point control represents the cFP-TC-120 FieldPoint module.

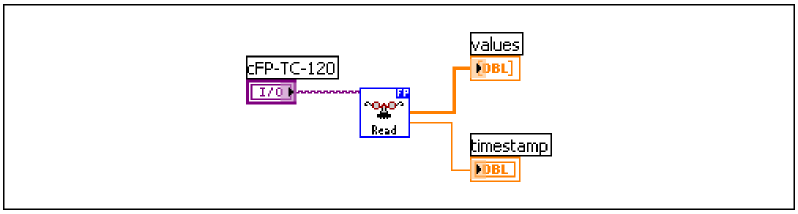


Figure 8-6. Measuring Temperature using FieldPoint

9 Measuring Current

This chapter describes how to measure current using DAQ devices, FieldPoint modules, and instruments.

Overview of Current Measurements

4 to 20 milliamp (4-20 mA) loops are commonly used in measurement systems. 4-20 mA loops couple a dynamic range with a live zero of 4 mA for open circuit detection in a system that does not produce sparks. Other advantages include a variety of compatible hardware, a long operating range up to 2,000 feet, and low cost. 4-20 mA loops have a variety of uses, including digital communications, control applications, and reading remote sensors.

The purpose of the 4-20 mA current loop is for the sensor to transmit a signal in the form of a current. In Figure 9-1, the Level Sensor and Remote Sensor Electronics are typically built into a single unit. An external 24 VDC supply powers the sensor. The sensor regulates the current, which represents the value of what the sensor measures, in this case, the fluid level in a tank.

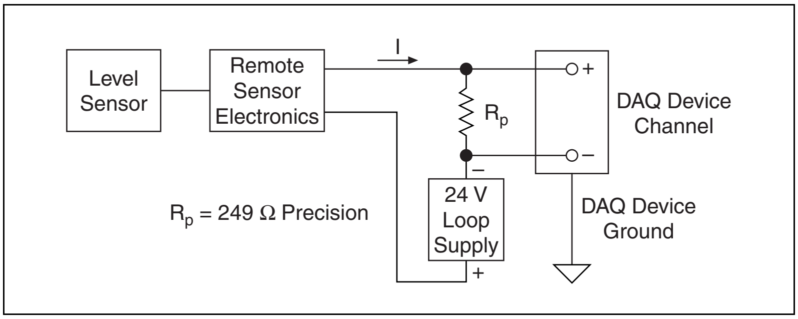


Figure 9-1. Current Loop Wiring

The DAQ device reads the voltage drop across the 249 Ω resistor R_p. Ohm's Law derives the current:

$$I_{(mA)} = \frac{V_{(Volts)}}{R_{p(Kohms)}}$$

Because the current is 4-20 mA and R_p is 249 Ω, V ranges from 0.996 V to 4.98 V, which is within the range that DAQ devices can read. Although the equation is useful for calculating the current, the current typically represents a physical quantity you want to measure. In Figure 9-2, the tank level measures 0 to 50 feet. 4 mA represents 0 feet, and 20 mA represents 50 feet. *L* is the tank level, and *I* is the current.

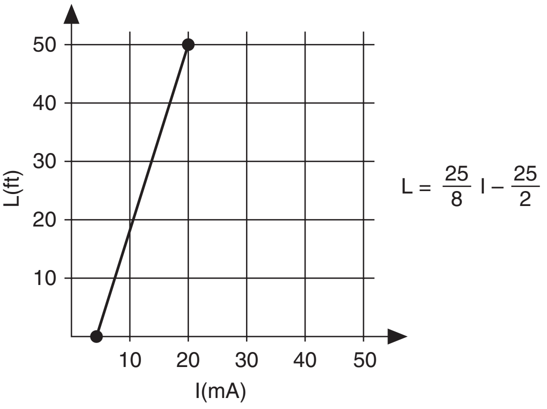


Figure 9-2. Linear Relationship between Tank Level and Current

Using the Ohm's Law equation and substituting 0.249 for the value of R_p, you can derive *L* in terms of measured voltage:

$$L = \frac{25 \times V}{8 \times 0.249} - \frac{25}{2}$$

Using NI-DAQ VIs to Measure Current

Figure 9-3 shows a DAQ system for measuring current to read the fluid level in a tank.

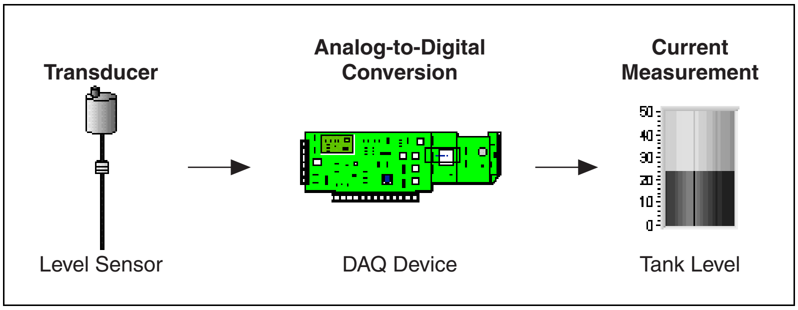


Figure 9-3. DAQ System for Current

Because multifunction DAQ (MIO) devices cannot directly measure current, you must use a precision resistor in series with the current loop circuit to read voltage, as shown in the current loop wiring diagram in Figure 9-1.

Traditional NI-DAQ Method

The block diagram in Figure 9-4 uses Traditional NI-DAQ VIs to implement the Ohm’s Law equation.

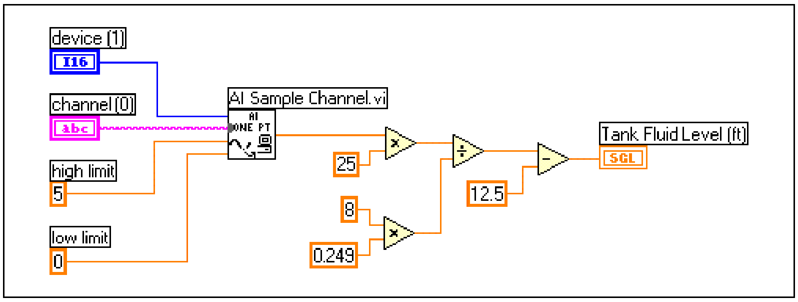


Figure 9-4. Measuring Fluid Level Without DAQ Named Channels

Alternatively, you can configure a DAQ Named Channel to handle this scaling as shown in Figure 9-5.

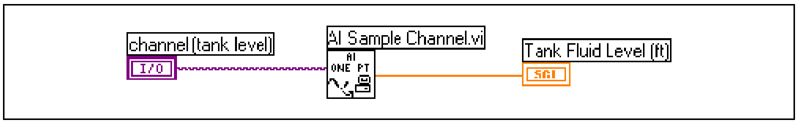


Figure 9-5. Measuring Fluid Level Using DAQ Named Channels

Measuring Current with Instruments

Figure 9-6 shows a measurement system that uses a stand-alone instrument to measure current. The stand-alone instrument also could be an instrument that plugs directly into a PC.

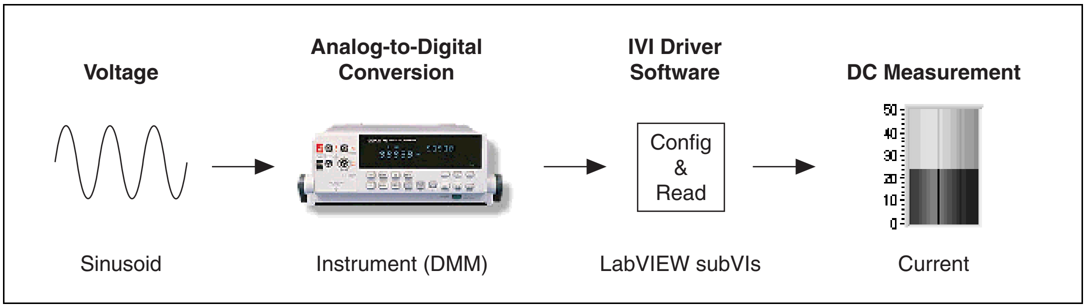


Figure 9-6. Instrument Control System for Measuring Current

The block diagram in Figure 9-7 uses the IVI class driver VIs to measure current. The IviDmm Initialize VI uses a logical name to create a session and initialize the instrument. The IviDmm Configure Measurement VI configures the measurement for current. The IviDmm Read VI takes the measurement, and the IviDmm Close VI closes the session.

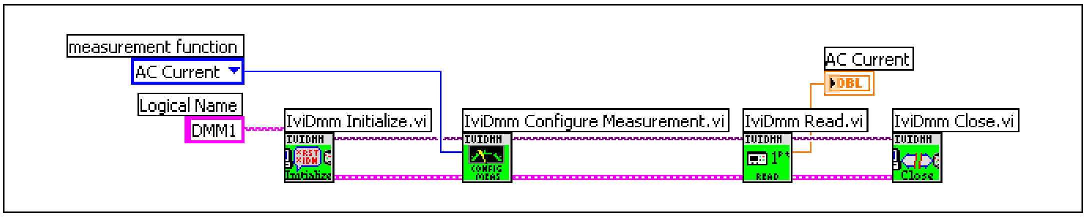


Figure 9-7. Measuring Current Using an Instrument

Using FieldPoint VIs to Measure Current

Figure 9-8 shows a FieldPoint system for measuring current.

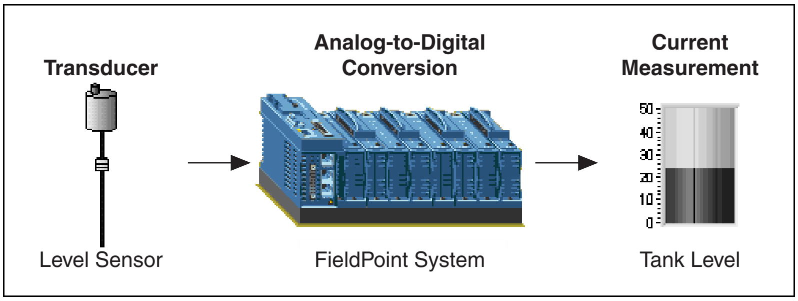


Figure 9-8. FieldPoint System for Measuring Current

The block diagram in Figure 9-9 uses a FieldPoint VI to measure current. In this example, the FieldPoint I/O Point control represents the cFP-AI-100 FieldPoint module.

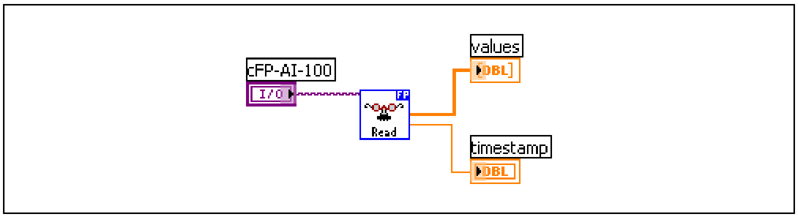


Figure 9-9. Measuring Current Using FieldPoint

10

Measuring Strain

This chapter describes how to measure strain using DAQ devices and FieldPoint modules.

Overview of Strain Measurements

Strain (e) is the amount of deformation of a body as a result of an applied force. Specifically, strain is defined as the fractional change in length, as shown in Figure 10-1.

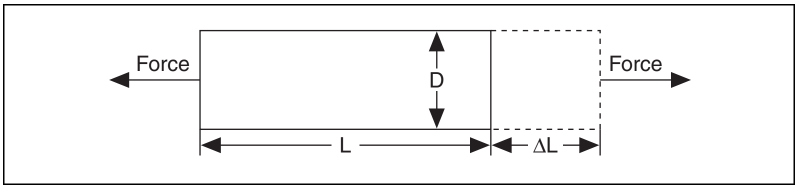


Figure 10-1. Strain (e)

Strain can be positive (tensile) or negative (compressive). Although dimensionless, strain is sometimes expressed in units such as in./in. or mm/mm. In practice, the magnitude of measured strain is very small. Therefore, strain is often expressed as microstrain (μe).

When a uniaxial force strains a bar as in Figure 10-1, a phenomenon known as Poisson Strain causes the girth of the bar, D, to contract in the transverse (perpendicular) direction. The Poisson's Ratio of a material indicates the magnitude of this transverse contraction. The Poisson's Ratio of a material is the negative ratio of the strain in the transverse direction (perpendicular to the force) to the strain in the axial direction (parallel to the force). For example, Poisson's Ratio for steel ranges from 0.25 to 0.30.

To measure strain, you typically use a strain gage with signal conditioning. Strain gages are thin conductors attached to the material to stress. Strain gages return varying voltages in response to stress or vibrations in materials. Resistance changes in parts of the strain gage to indicate deformation of the material. Strain gages require excitation (generally voltage excitation) and linearization of the voltage measurements.

Depending on the strain gage configuration, another requirement for using strain gages with signal conditioning is a configuration of resistors. As shown in Figure 10-2, the resistance from the strain gages combined with signal conditioning hardware form a diamond-shaped configuration of resistors, known as a Wheatstone bridge. When you apply a voltage to the bridge, the differential voltage (V_m) varies as the resistor values in the bridge change. The strain gage usually supplies the resistors that change value with strain.

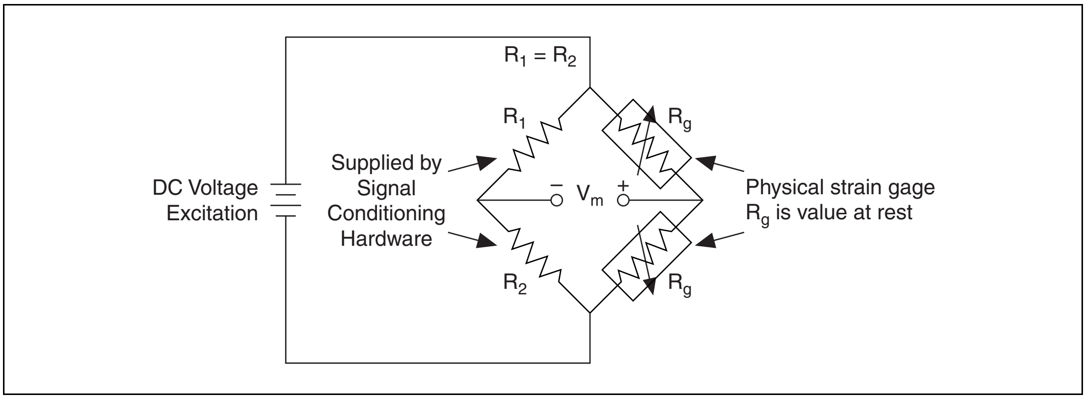

$R_1 = R_2$
R_1
R_g
Supplied by Signal Conditioning Hardware
DC Voltage Excitation
– V_m +
Physical strain gage R_g is value at rest
R_2
R_g

Figure 10-2. Half-Bridge Wheatstone Strain Gauge

Strain gages come in full-bridge, half-bridge, and quarter-bridge configurations. For a full-bridge strain gage, the four resistors of the Wheatstone bridge are physically located on the strain gage itself. For a half-bridge strain gage, the strain gage supplies two resistors for the Wheatstone bridge, and the signal conditioning hardware supplies the other two resistors, as shown in Figure 10-2. For a quarter-bridge strain gage, the strain gage supplies only one of the four resistors for a Wheatstone bridge.

The National Instruments SCXI-1520 module is a dedicated strain measuring module with software-configurable bridge-completion, excitation, resistance shunt switches, filter, and gain on each of the eight channels.

The National Instruments SCXI-1121 module and the National Instruments SCXI-1122 module are commonly used with strain gages because they include voltage or current excitation and internal Wheatstone bridge completion circuits. As an alternative to SCXI modules, you can use the signal conditioning device SC-2043SG, which is designed specifically for strain gage measurements. Refer to the National Instruments catalog for more information about this device.

You can set up a SCXI module to amplify strain gage signals or to filter noise from signals. Refer to the *Getting Started with SCXI* manual for the necessary hardware configuration and for information about configuring the excitation level, gain, and filter settings.

Using NI-DAQmx VIs to Measure Strain

The block diagram in Figure 10-3 uses the NI-DAQmx **Task Name Constant** to measure strain. The DAQ Assistant configures the `MyStrainTask`, which contains strain information, such as bridge configuration, excitation voltage, gage factor, and so on. The DAQmx Read VI measures strain and graphs the data. By using a NI-DAQmx **Task Name Constant** with the DAQ Assistant, you can configure the task and make edits without changing code on the block diagram.

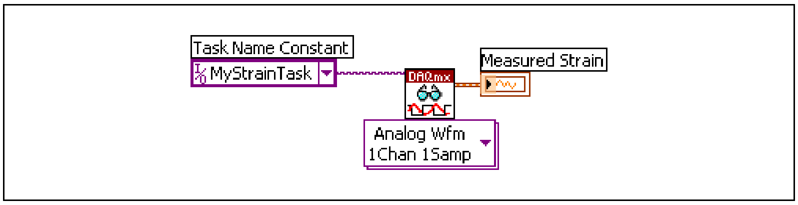


Figure 10-3. Using a Task I/O Constant to Measure Strain

Using FieldPoint VIs to Measure Strain

The block diagram in Figure 10-4 uses a FieldPoint VI to measure strain. In this example, the FieldPoint I/O Point control represents the cFP-SG-140 FieldPoint module.

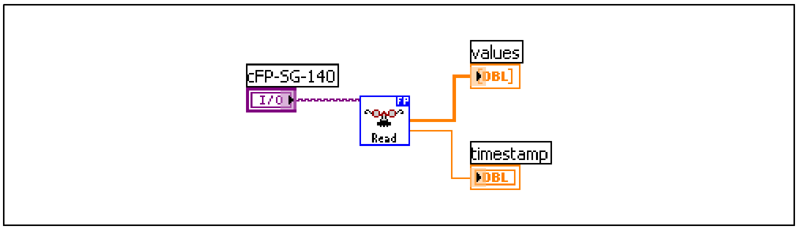


Figure 10-4. Measuring Strain Using FieldPoint

11

Measuring Resistance

This chapter describes how to measure resistance using instruments.

Overview of Resistance Measurements

Resistance is the opposition to the flow of electric current. One ohm (Ω) is the resistance through which one volt of electric force causes one ampere to flow.

Two common methods for measuring resistance are a 2-wire method and a 4-wire method. Both methods send a current through a resistor, and a measurement device measures the voltage drop from the signal before and after it crosses the resistor. Use the following equation to calculate resistance:

$$R = \frac{V}{I}$$

where R = resistance, V = voltage, and I = current.

2-Wire Resistance

Use the 2-wire method as shown in Figure 11-1 to measure resistances greater than 100 Ω.

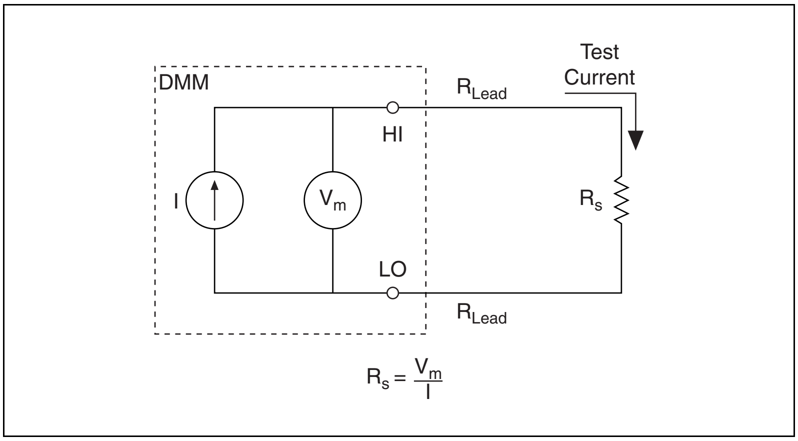


Figure 11-1. 2-Wire Resistance Method

The excitation current flows through the leads and the unknown resistance, R_S. The device measures the voltage across the resistance through the same set of leads and computes the resistance accordingly. The lead resistance, R_{Lead}, introduces errors in the 2-wire measurements when you measure lower resistances. Because a voltage drop exists across the lead resistance equal to $I \times R_L$, the voltage the device measures is not exactly the same as the voltage across the resistance, R_S. Because typical lead resistances lie in the range of 0.01–1, accurate 2-wire resistance measurements are difficult to obtain if R_S is below 100 Ω.

4-Wire Resistance

Use the 4-wire resistance method as shown in Figure 11-2 to measure resistances of less than 100 Ω because it is more accurate than the 2-wire method.

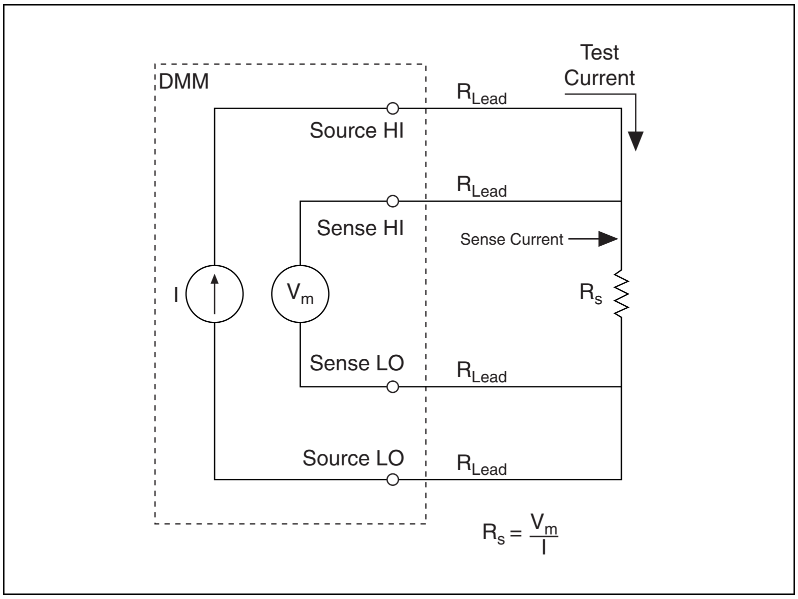


Figure 11-2. 4-Wire Resistance Method

The 4-wire method uses four leads, one pair for the injected current (the test current) and the other pair for sensing the voltage across the resistor (the sense current). Because no current flows in the sense lead, a device measures only the voltage developed across the resistance. Thus, a 4-wire resistance eliminates errors test lead and contact resistance cause.

Using DMMs to Measure Resistance

Figure 11-3 shows a measurement system to measure resistance.

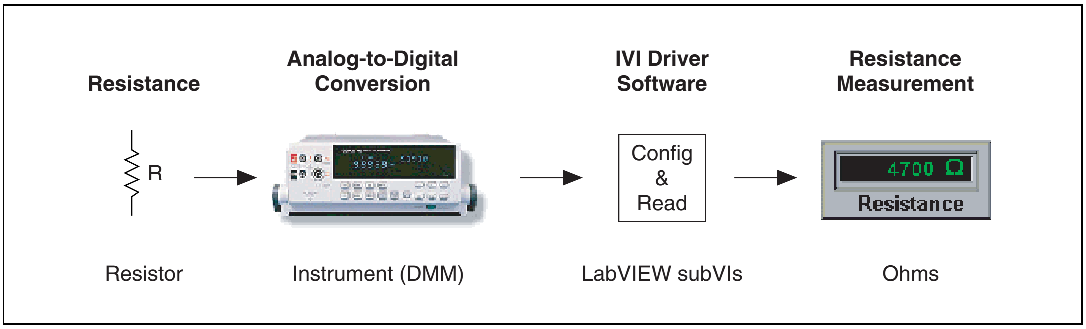


Figure 11-3. Instrument Control System for Resistance

The block diagram in Figure 11-4 uses the IVI class driver VIs to measure resistance. The IviDmm Initialize VI uses a logical name to create a session and initialize the instrument. The IviDmm Configure Measurement VI configures the measurement for resistance. The IviDmm Read VI takes the measurement, and the IviDmm Close VI closes the session.

Notice that this block diagram is similar to Figure 6-10, *Measuring DC Voltage Using IVI Class Driver VIs*. The difference is that the block diagram in Figure 11-4 uses the 2-wire resistance for the measurement function.

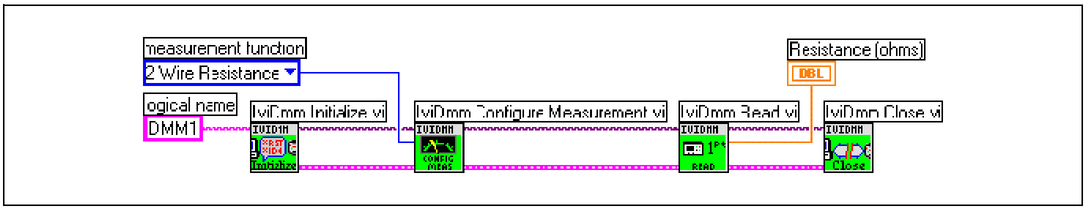


Figure 11-4. Measuring Resistance Using an Instrument

12

Generating Voltage

This chapter describes how to generate voltage using DAQ devices and instruments.

Overview of Generating Voltage

You can generate a single DC signal or a time-varying, also called a buffered, signal.

Single-Point Analog Output

When the signal level at the output is more important than the rate at which the output value changes, you need to generate a steady DC value. You can use single-point analog output VIs to produce this type of output. With single-point analog output, you must call one of the VIs that produces a single update, or a single value change, any time you want to change the value on an analog output channel. Therefore, you can change the output value only as fast as LabVIEW calls the VIs. This technique is called software timing. Use software timing if you do not need high-speed generation or the most accurate timing. Refer to the *Hardware versus Software Timing* section of Chapter 4, *Measurement Fundamentals*, for more information about software timing.

Buffered Analog Output

Sometimes the rate at which the output value changes is just as important as the signal level, as in waveform generation or buffered analog output. For example, you might want a DAQ device to act as a function generator. You can accomplish this by using a VI that generates one cycle of a sine wave, such as the Sine Generation VI, stores one cycle of sine wave data in a waveform, and programs the DAQ device to generate the values continuously from the waveform one point at a time at a specified rate. You can use circular-buffered analog output to generate a continually changing waveform. For example, you might have a large file stored on disk that contains data you want to output. If the computer cannot store the entire waveform in a single buffer, you must continually load new data into the buffer during the generation.

Connecting Analog Output Signals

Signal connections vary depending on the device, connector block, and signal conditioning module. For E Series devices, the analog output signals are AO0, AO1, and AO GND. AO0 is the voltage output signal for analog output channel 0. AO1 is the voltage output signal for analog output channel 1. AO GND is the ground reference signal for both analog output channels and the external reference signal. Figure 12-1 shows how to make analog output connections for an NI device.

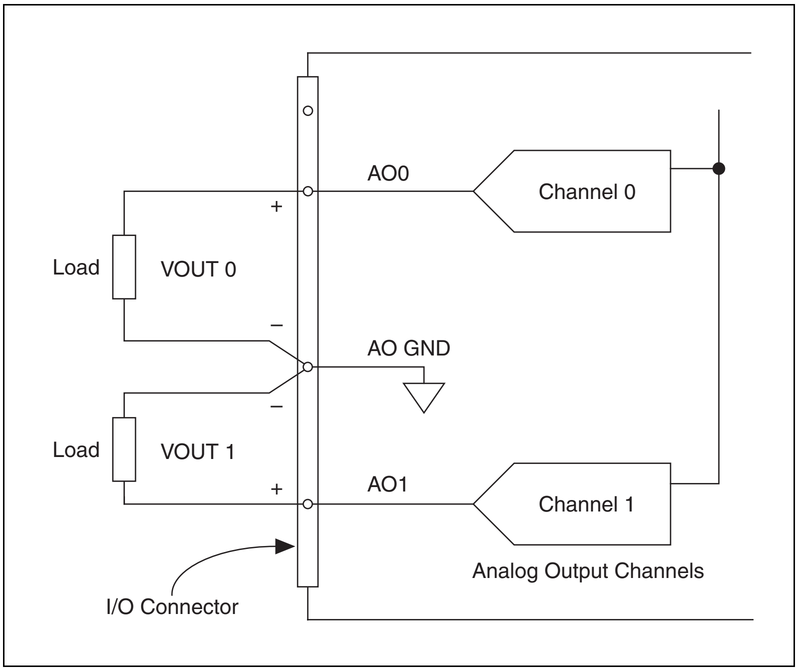


Figure 12-1. Analog Output Connections

Refer to the device documentation for information about specific terminals.

Using Traditional NI-DAQ VIs for Single-Point Updates

You can use Traditional NI-DAQ VIs for immediate single-point updates and for multiple immediate updates.

Immediate Updates

The simplest way to program single-point updates with Traditional NI-DAQ VIs is to use the AO Update Channels VI, which writes values to one or more output channels on the output DAQ device.

When you wire an array of values to the AO Update VI, the first element in the array corresponds to the first entry in the channel string, and the second array element corresponds to the second entry in the channel string. If you use a DAQ Named Channel in a channel string, the **values** input is relative to the physical units you specify in the DAQ Channel Wizard, a utility that guides you through naming and configuring DAQ analog and digital signals. Otherwise, the **values** input represents volts. Because the AO Update Channels VI is an Easy Analog Output VI, it includes built-in error handling.

Refer to the Generate 1 Point on 1 Channel VI in the `examples\daq\anlogout\anlogout.llb` for an example of generating one value for one channel.

If you want more control over the limit settings for each channel and more control over when you can check for errors, use the AO Write One Update VI, an Intermediate VI. The **iteration** input optimizes the execution of this VI if you place it in a loop.

Multiple Immediate Updates

Refer to the Write N Updates VI in the `examples\daq\anlogout\anlogout.llb` for an example of performing multiple updates. The Write N Updates VI is similar to the AO Write One Update VI except that the While Loop in the Write N Updates VI runs the subVI repeatedly until the error status or the **stop** Boolean value is `TRUE`. You can use the AO Update Channels VI in a loop but doing so is inefficient because the VI configures the device each time the VI runs. The AO Write One Update VI configures the device only when the value of the **iteration** input is `0`.

The Write N Updates VI illustrates an immediate, software-timed analog output VI application in which software timing in a loop controls the update rate. A good reason to use immediate, software-timed output is that the application calculates or processes output values one at a time. However, remember that software timing is not as accurate as or as quick as hardware-timed analog output.

Using Traditional NI-DAQ VIs for Waveform Generation

You can use Traditional NI-DAQ VIs to generate single-buffered analog output and circular-buffered analog output.

Single-Buffered Analog Output

Use the AO Generate Waveforms VI, an Easy Analog Output VI, to generate single-buffered analog outputs. The AO Generate Waveforms VI writes an array of output values to the analog output channels at a rate you specify in **update rate**. For example, if **channels** consists of two channels, and the **waveforms** array consists of waveform data for the two channels, the AO Generate Waveforms VI writes values from the waveform array to the corresponding channels at every update interval. The VI stops after it writes all the values in the array to the channels. The signal level on the output channels maintains the value of the final value row in the waveform array until another value is generated. If you use DAQ Named Channels in the **channels** input, **waveforms** is relative to the units you specify in the DAQ Channel Wizard. Otherwise, **waveforms** represents volts.

You also can use the AO Waveform Gen VI to generate a single-buffered analog output. Use the generation count input of the AO Waveform Gen VI to generate data once, several times, or continuously.

Refer to the Generate N Updates VI in the `examples\daq\anlogout\anlogout.llb` for an example of the AO Waveform Gen VI. Placing the AO Waveform Gen VI in a loop and wiring the iteration terminal of the loop to the iteration input of the AO Waveform Gen VI optimizes the execution of the Generate N Updates VI. When the iteration is `0`, LabVIEW configures the analog output channels appropriately. If the iteration is greater than `0`, LabVIEW uses the existing configuration, which improves performance. With the AO Waveform Gen VI, you also can specify the limit settings input for each analog output channel.

To gain even more control over an analog output application, use the Intermediate Traditional DAQ VIs as shown in Figure 12-2.

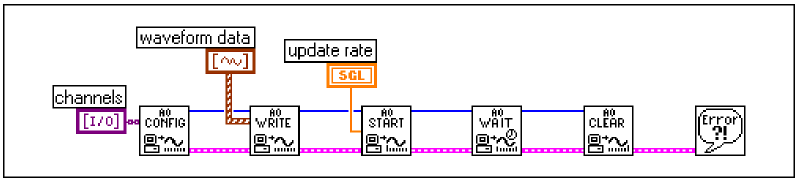


Figure 12-2. Waveform Generation Using Intermediate VIs

With the Intermediate Traditional DAQ VIs, you can set up an alternate update clock source, such as an external clock or a clock signal coming from another device, or you can return the update rate. The AO Config VI configures the channels you specify for analog output. The AO Write VI places the data in a buffer. The AO Start VI begins the actual generation at the **update rate**. The AO Wait VI waits until the waveform generation completes. The AO Clear VI clears the analog channels.

Refer to the Generate Continuous Sinewave VI in the `examples\daq\anlogout\anlogout.llb` for an example of continually generating a sine waveform through the channel you specify.

Circular-Buffered Analog Output

When the waveform data is too large to fit in a memory buffer or is constantly changing, use a circular buffer to output the data. You can use the Easy Analog Output VIs in a loop to create a circular-buffered output, but this sacrifices efficiency because Easy VIs configure, allocate, and deallocate a buffer every time they execute, which causes time gaps between the data output.

Refer to the AO Continuous Gen VI for an example of using the Intermediate Traditional DAQ VIs to see one way to perform circular-buffered analog output. The AO Continuous Gen VI is more efficient than the Easy Analog Output VIs because it configures and allocates a buffer when the **iteration** input is `0` and deallocates the buffer when the **clear generation** input is `TRUE`. With the AO Continuous Gen VI, you can configure the size of the data buffer and the limit settings of each channel.

Refer to the Continuous Generation VI in the `examples\daq\anlogout\anlogout.llb` for an example of using the AO Continuous Gen VI. In this example, the data completely fills the buffer on the first iteration. On subsequent iterations, the VI writes new data into half of the buffer while the other half of the buffer continues to output data.

To gain more control over an analog output application, use the Traditional NI-DAQ Intermediate VIs as shown in Figure 12-3.

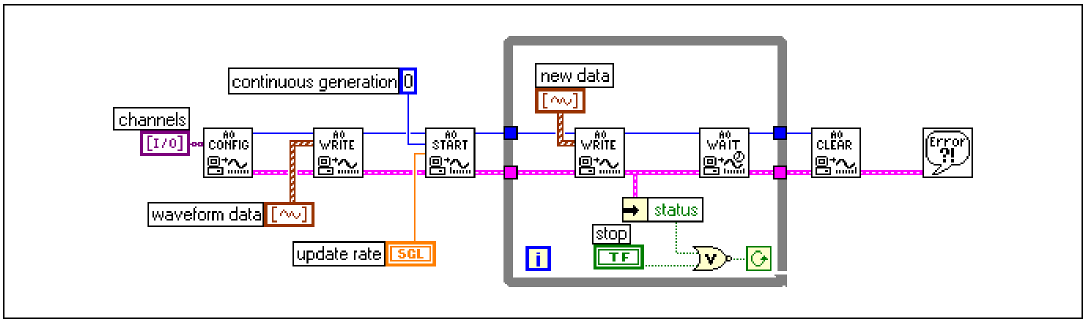


Figure 12-3. Circular Buffered Waveform Generation Using Traditional NI-DAQ Intermediate VIs

With the Traditional NI-DAQ Intermediate VIs, you can set up an alternate update clock source, such as an external clock or a clock signal coming from another device, and you can monitor the update rate the VI actually uses. The AO Config VI configures the channels you specify for analog output. The AO Write VI places the data in a buffer. The AO Start VI begins the actual generation at the **update rate**. The AO Write VI in the While Loop writes new data to the buffer until you click the **Stop** button. The AO Clear VI clears the analog channels.

Refer to `examples\daq\anlogout\anlogout.llb` for an example of the Function Generator VI. The Function Generator VI changes the output waveform on-the-fly and responds to changing signal types (sine or square), amplitude, offset, update rate, and phase settings on the front panel.

Using NI-DAQmx VIs to Generate Voltage

You can use NI-DAQmx VIs to generate voltage.

The block diagram in Figure 12-4 uses NI-DAQmx VIs to generate a sine wave on an analog output channel. The Sine Waveform VI generates a sine wave with a frequency of 10 Hz and an amplitude of 1 V. The DAQmx Write VI writes the sine wave data to the specified physical channel. The DAQmx Timing VI provides the timing information necessary for voltage generation. The DAQmx Wait Until Done VI waits for the sine wave generation to complete. Without the DAQmx Wait Until Done VI, the voltage generation might prematurely stop, which could lead to data loss.

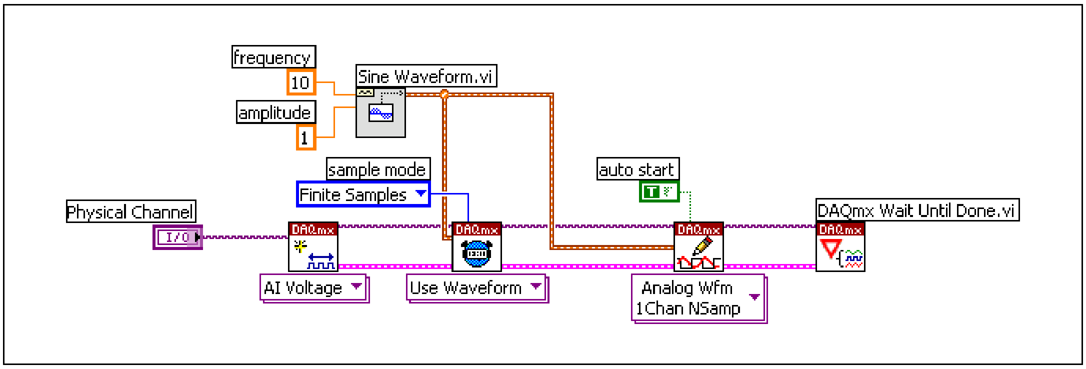


Figure 12-4. Using NI-DAQmx VIs to Generate a Sine Wave

Generating Voltage with Instruments

The block diagram in Figure 12-5 uses the IVI class driver VIs to generate a sine waveform with a frequency of 5 kHz and an amplitude of 2 volts. The IviFgen Initialize VI uses a logical name to create an IVI instrument driver session to the device. The IviFgen Configure Standard Waveform [STD] VI specifies the frequency and amplitude for the waveform. The IviFgen Initiate Generation VI sends the waveform configuration to the instrument and generates the waveform.

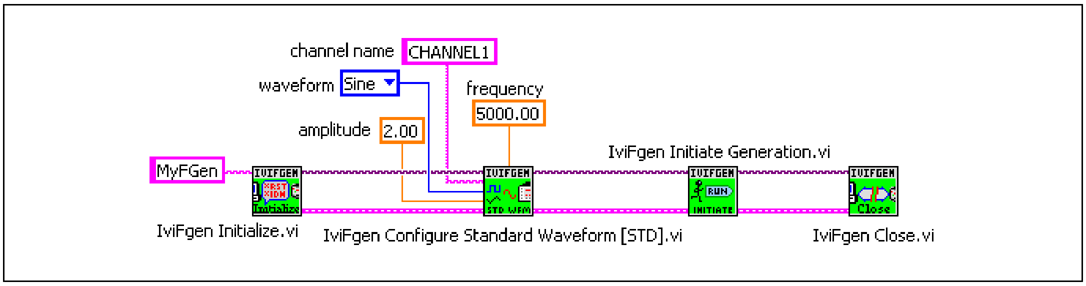


Figure 12-5. Using IVI VIs to Generate Voltage

13

Measuring Analog Frequency

This chapter describes how to measure analog frequency using DAQ devices and instruments.

Using NI-DAQ VIs to Measure Analog Frequency

You can use Traditional NI-DAQ and NI-DAQmx to measure analog frequency.

The Nyquist Theorem states that the highest frequency you can accurately represent is equal to half the sampling rate. This means that if you want to measure the frequency of a 100 Hz signal, you need a sampling rate of at least 200 S/s. In practice, use sampling rates of five to 10 times the expected frequencies.

In addition to sample rate, you need to determine the number of samples to acquire. You must sample a minimum of three cycles. In practice, however, acquire 10 or more cycles. For example, you need to collect at least 15 samples, or points, if you use a sampling rate of 500 S/s to measure the frequency of a 100 Hz signal. Because you are sampling about five times faster than the signal frequency, you sample about five points per cycle of the signal. Because you need data from three cycles, 5 points × 3 cycles = 15 points.

The number of points you collect determines the number of frequency bins that the data falls into. The size of each bin is the sampling rate divided by the number of points you collect. For example, if you sample at 500 S/s and collect 100 points, you have bins at 5 Hz intervals.

Traditional NI-DAQ Method

The block diagram in Figure 13-1 uses Traditional NI-DAQ VIs to measure analog frequency. The AI Acquire Waveform VI reads samples from the **channel** input. According to the Nyquist frequency, the **number of samples** should be no more than 500 if the **sample rate** is 1,000 samples per second. The Extract Single Tone Information VI returns the frequency reading.

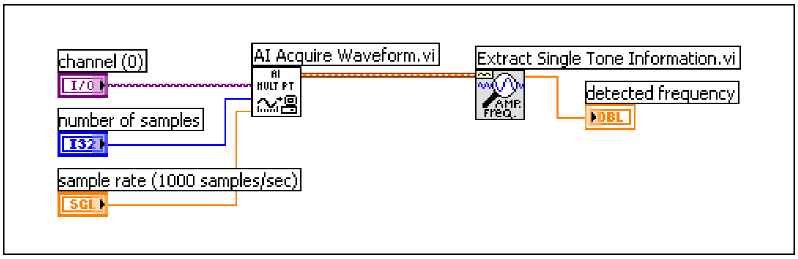


Figure 13-1. Measuring Frequency with Traditional NI-DAQ VIs

NI-DAQmx Method

The block diagram in Figure 13-2 uses NI-DAQmx VIs to measure the analog frequency of a Waveform. The DAQmx Create Virtual Channel VI creates a virtual channel that acquires a voltage signal. The DAQmx Timing VI is set to Sample Clock with the sample mode set to Finite. Samples per Channel and Rate determine how many samples per channel to acquire and at what rate. This example returns 100 samples at a rate of 500 samples per second, so the acquisition takes 1/5 of a second and terminates. The DAQmx Read VI measures the 100 voltage samples and sends the waveform to the Extract Single Tone Information VI, which returns the frequency reading.

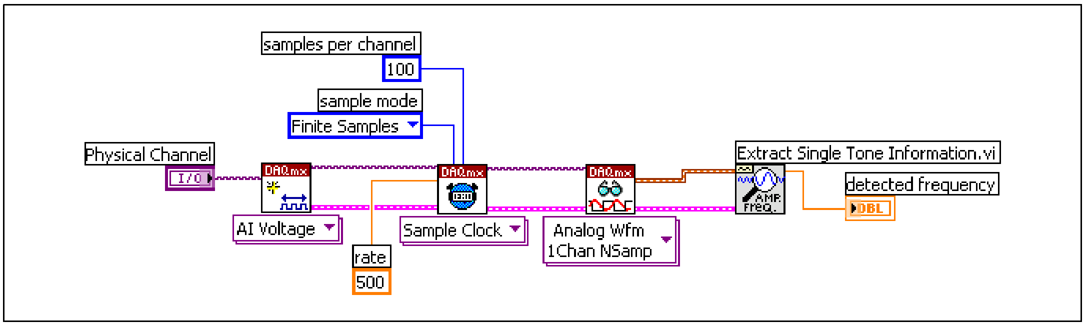


Figure 13-2. Measuring Analog Frequency with NI-DAQmx VIs

To acquire frequency readings from multiple channels, select multiple channels in the **Physical Channel** I/O control, configure the DAQmx Read VI to read multiple samples from multiple channels, and update the Extract Single Tone Information VI to return an array of detected frequencies.

Measuring Frequency Using Instruments

The block diagram in Figure 13-3 uses the IVI class driver VIs to measure frequency. Because frequency measurement is inherent to the instrument, the VI does not calculate the frequency value. The instrument returns the frequency value.

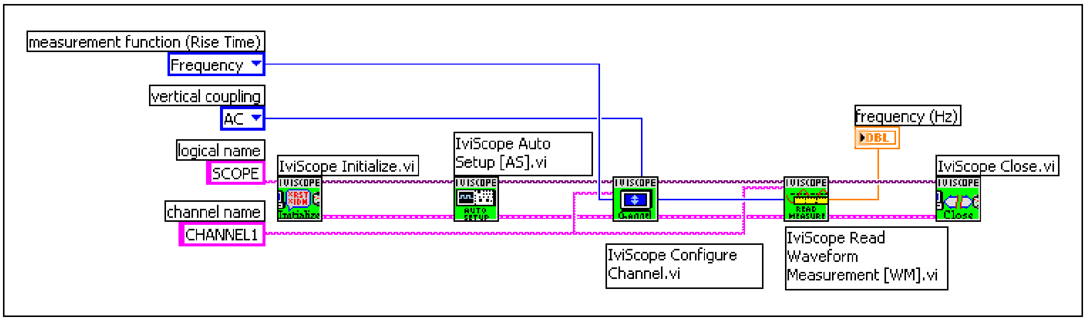

measurement function (Rise Time)
Frequency
vertical coupling
AC
logical name
SCOPE
channel name
CHANNEL1
IviScope Initialize.vi
IviScope Auto Setup [AS].vi
IviScope Configure Channel.vi
frequency (Hz)
DBL
IviScope Read Waveform Measurement [WM].vi
IviScope Close.vi

Figure 13-3. Measuring Frequency Using an Instrument

The IviScope Initialize VI uses a logical name to create a session and initialize the instrument. The IVIScope Auto Setup [AS] configures the default for the scope. The IviScope Configure Channel VI configures the measurement for frequency. The IviScope Read Waveform Measurement [WM] VI takes the measurement, and the IviScope Close VI closes the session.

Notice that the IVI VIs in Figure 13-3 are like those in Figure 7-9, *Measuring Peak-to-Peak Voltage Using IVI Class Driver VIs*. The only difference is the configuration of the measurement function.

Measuring Frequency with Filtering

The Nyquist frequency is the bandwidth of the sampled signal and is equal to half the sampling frequency. Frequency components below the Nyquist frequency appear normally. Frequency components above the Nyquist frequency appear aliased between 0 and the Nyquist frequency. The aliased component is the absolute value of the difference between the actual component and the closest integer multiple of the sampling rate. For example, if you have a signal with a component at 800 Hz and you sample at 500 S/s, that component appears aliased at 200 Hz because

$$|800 - (2 \times 500)| = 200(\text{Hz})$$

One way to eliminate aliased components is to use an analog hardware filter before you digitize and analyze the frequency information. If you want to perform all the filtering in software, you must first sample at a rate fast enough to correctly represent the highest frequency component the signal contains. For example, with the highest component at 800 Hz, the minimum sample rate is 1,600 Hz, but you should use a sampling rate five to 10 times faster than 800 Hz. If the frequency you are trying to measure is around 100 Hz, you can use a lowpass Butterworth filter with a cutoff frequency (f_c) of 250 Hz to filter out frequencies above 250 Hz and pass frequencies below 250 Hz. Figure 13-4 shows a lowpass filter.

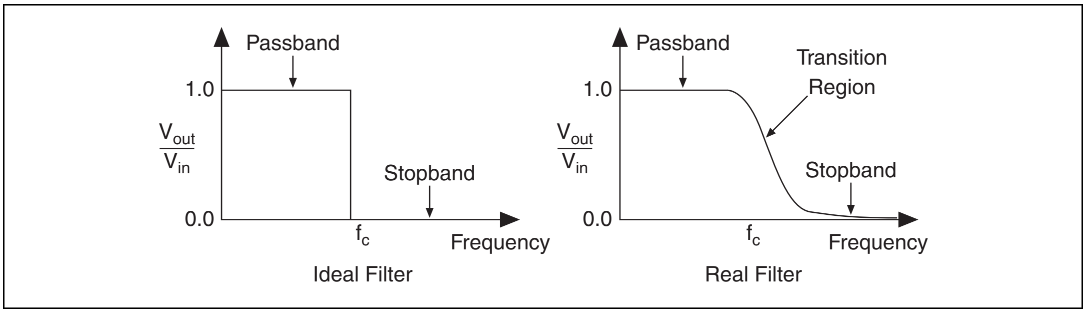


Figure 13-4. Lowpass Filter

The Ideal Filter in Figure 13-4 is optimal. All frequencies above the Nyquist frequency are rejected. The Real Filter in Figure 13-4 is what you might actually be able to accomplish with a Butterworth filter. The passband is where V_{out}/V_{in} is close to 1. The stopband occurs where V_{out}/V_{in} is close to 0. The frequencies gradually attenuate on the transition region between 1 and 0.

The block diagram in Figure 13-5 filters the signal before it measures the frequency.

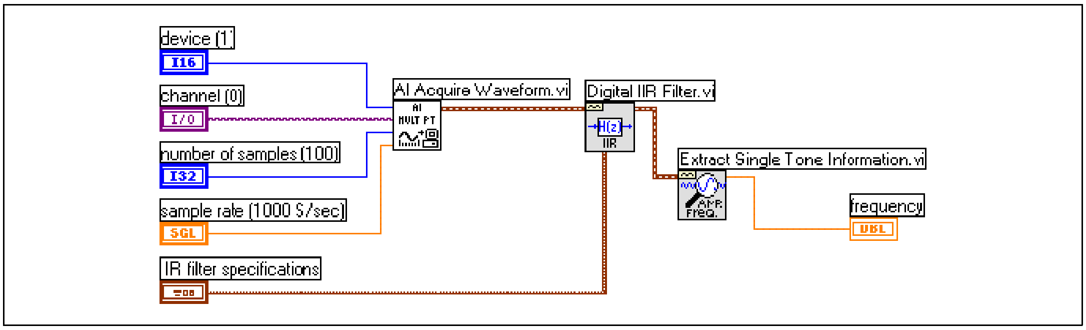


Figure 13-5. Measuring Frequency after Filtering

Notice the Digital IIR Filter VI and the **IIR filter specifications** control as shown in Figure 13-6. You use the **IIR filter specifications** control to select the design parameters for the filter.

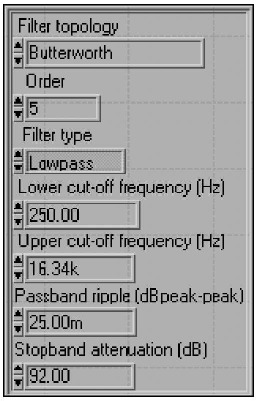

Filter topology
Butterworth
Order
5
Filter type
Lowpass
Lower cut-off frequency (Hz)
250.00
Upper cut-off frequency (Hz)
16.34k
Passband ripple (dBpeak-peak)
25.00m
Stopband attenuation (dB)
92.00

Figure 13-6. IIR Filter Specifications

In this example, the fifth-order lowpass Butterworth filter uses a cutoff frequency of 250 Hz. The order of a filter determines how steep the transition region is. A higher order yields a steeper transition. However, a lower order decreases computation time and error. In this example, the filter ignores the **Upper cut-off frequency**, **Passband ripple**, and **Stopband attenuation**. Refer to Chapter 4, *Digital Filtering*, of the *LabVIEW Analysis Concepts* manual, for more information about filtering.

14 Measuring Digital Pulse Width, Period, and Frequency

This chapter describes how to measure time using digital pulse width, period, and frequency using DAQ devices and counters.

Overview of Counters

Counters typically operate with TTL signals. Refer to the *Digital I/O* section of Chapter 4, *Measurement Fundamentals*, for more information about TTL signals.

Counters monitor the state of the signal and transition the signal from one state to another. A counter also can detect a rising edge, which is a transition from logic low to logic high, and a falling edge, which is a transition from logic high to logic low. The rise time and the fall time is the amount of time it takes for the rising edges and falling edges to occur, respectively. As defined by the specifications for a TLL signal, the transition must occur within 50 ns or less for a counter to detect the edge, as shown in Figure 14-1.

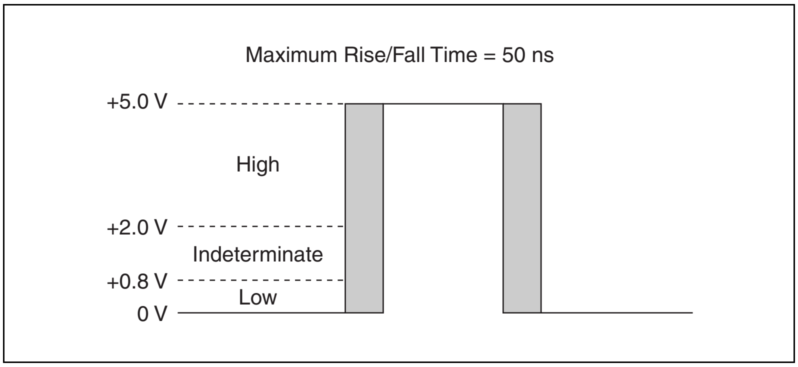


Figure 14-1. Detecting Rising/Falling Edges

Counter Parts

Figure 14-2 shows the main parts of a counter.

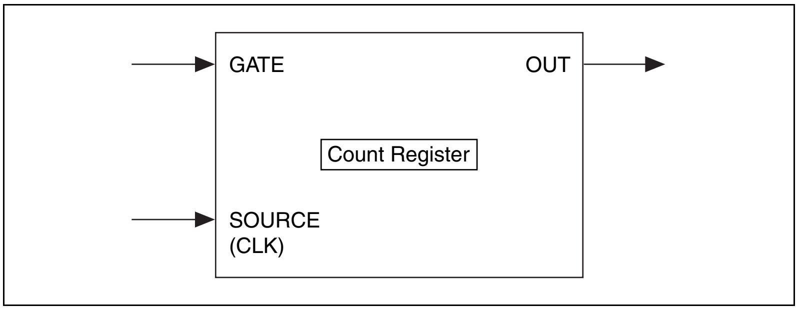


Figure 14-2. Counter Parts

A GATE input terminal controls when counting occurs. A GATE input is similar to a trigger because it starts or stops a count.

A SOURCE (CLK) input terminal is the timebase for a measurement or the signal to count.

A count register increments or decrements the number of edges to count. If the count register decrements, it counts down to zero. The count register size is the number of bits in the counter, and you calculate it as Count Register = $2^{\# \text{ of bits}}$.

An OUT signal terminal can output a pulse or a pulse train, which is a series of pulses.

Overview of Time Measurements

You can measure time using counters to determine the duration of an event or to determine the interval time between two events. For example, you can use this type of measurement to determine the time interval between two boxes on a conveyor belt. The event is an edge every time a box goes by a point, which prompts a digital signal to change in value

Time measurements consist of digital pulse width, period, and frequency. Pulse width measures the time between a rising edge and a falling edge or a falling edge and a rising edge. Period measures the time between consecutive rising edges or falling edges. Frequency is the inverse of the period. Figure 14-3 shows the difference between period and pulse width measurements.

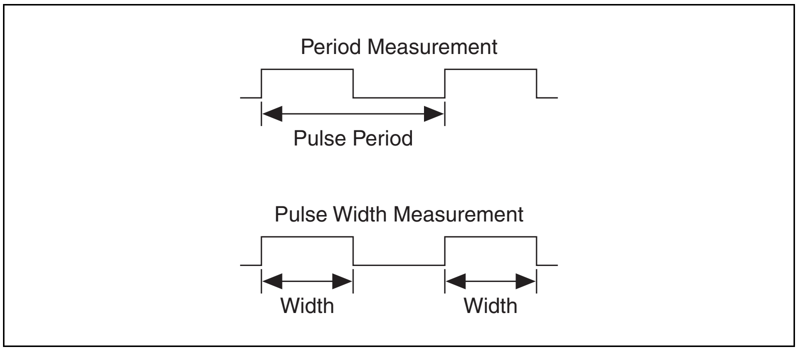


Figure 14-3. Period and Pulse Width Measurements

Use the following equation to calculate period and pulse width:

$$\text{Period or Pulse Width (in seconds)} = \frac{\text{Count}}{\text{Counter Timebase Rate}}$$

where Count is the number of the Counter Timebase Rate ticks that elapses during one period or pulse width of measuring the input signal.

Frequency is the inverse of the period of a signal. Use the following equation to calculate frequency:

$$\text{Frequency (in Hz)} = \frac{\text{Counter Timebase Rate}}{\text{Count}}$$

You can take measurements in terms of frequency and time when the counter timebase rate is a known frequency. If the counter timebase rate is unknown, you can take measurements only in terms of counter timebase ticks. The counter timebase rate might be unknown if you use an external signal with an unknown counter timebase frequency.

Quantization Error

Quantization error is the inherent uncertainty in digitizing an analog value as a result of the finite resolution of the conversion process. Quantization error depends on the number of bits in the converter, along with its errors, noise, and nonlinearities. Quantization errors occur as a result of phase differences between the input signal and counter timebase and can be different depending on the rate of the input signal and the measurement method you use.

Figure 14-4 shows three possible results when you measure time using counters.

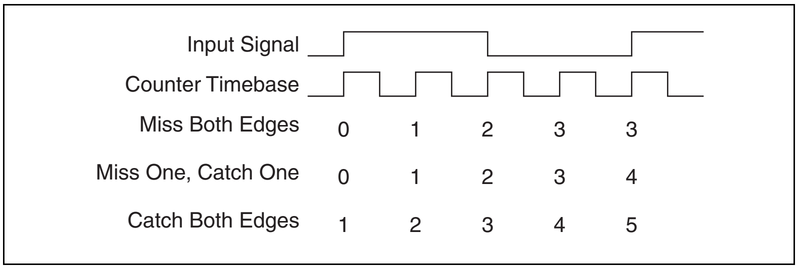

Input Signal
Counter Timebase
Miss Both Edges 0 1 2 3 3
Miss One, Catch One 0 1 2 3 4
Catch Both Edges 1 2 3 4 5

Figure 14-4. Quantization Error with Counters

- **Miss Both Edges**—The counter misses the first rising edge and the last rising edge of the counter timebase, which happens if the input signal transitions right before the first counter timebase edge and right after the last counter timebase edge. The result is a count of one less than the expected value.
- **Miss One, Catch One**—The counter recognizes only the first rising edge or the last rising edge of the counter timebase. The result is the expected value.
- **Catch Both Edges**—The counter recognizes the first rising edge and the last rising edge of the counter timebase. The result is a count of one more than the expected value.

For example, if the counter timebase rate is 20 MHz and the frequency of the input signal is 5 MHz, the count could be 3, 4, or 5 as a result of a quantization error. The 20 MHz counter timebase with the counts of 3, 4, or 5 corresponds to a measured frequency of 6.67 MHz, 5 MHz, or 4 MHz, resulting in a quantization error of as much as 33%.

Quantization Error with Counter Time Measurements

Use the following equation to calculate quantization error for time measurements that use a single counter:

$$\mathrm{Err}_{\mathrm{Quantization}} = \frac{\text{Actual Frequency}}{(\text{Counter Timebase Rate} - \text{Actual Frequency})}$$

You can reduce the quantization error for a time measurement by increasing the counter timebase rate. Table 14-1 lists the quantization error for various counter timebase rates and input signal frequencies.

Table 14-1. Quantization Error with Counter Time Measurements

Actual Frequency of Input Signal	Counter Timebase Rate	Quantization Error
10 Hz	100 kHz	0.01%
100 Hz	100 kHz	0.10%
1 kHz	100 kHz	1.01%
10 kHz	20 MHz	0.05%
100 kHz	20 MHz	0.50%
1 MHz	20 MHz	5.26%

Two Counter Measurement Method

You can measure period and frequency using one or two counters. For most applications, one counter is sufficient and uses fewer system resources. You might want to use the high-frequency two counter measurement method or the large-range two counter measurement method if you have a high frequency or a widely varying signal.

High-Frequency Two Counter Measurement Method

Use the high-frequency measurement method if you measure a digital frequency or period of a signal with a high frequency component. This method uses a second counter, as shown in Figure 14-5, to generate a pulse train with a known period, also called the measurement time.

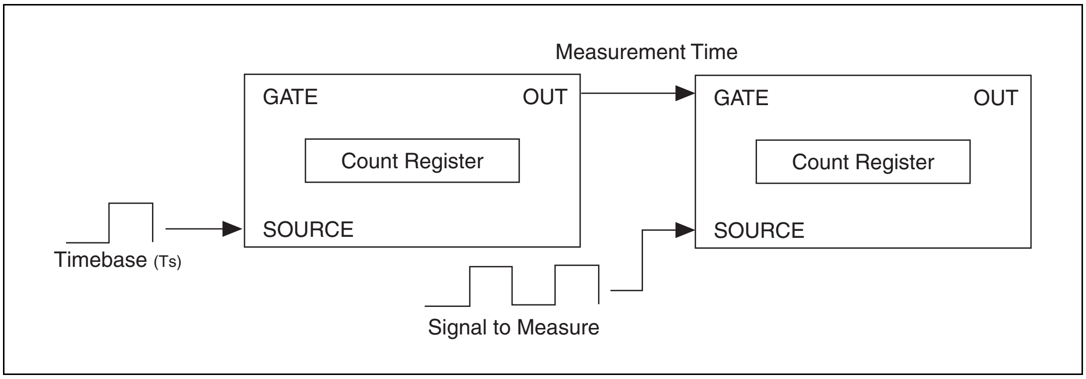


Figure 14-5. High-Frequency Two Counter Measurement Method

To reduce quantization error, the measurement time is larger than the period of the input signal, but the measurement time must be small enough to keep the count register from rolling over. The measurement counter counts the number of input signal periods that occur during the measurement time, averages the results, and returns the average value in the NI-DAQmx Read VI.

Use the following equations to calculate the average value:

$$\text{Period (in seconds)} = \frac{\text{Measurement Time}}{\text{Number of Periods Counted}}$$

$$\text{Frequency (Hz)} = \frac{\text{Number of Periods Counted}}{\text{Measurement Time}}$$

Quantization Error with High-Frequency Two Counter Measurement Method

Use the following equations to calculate the quantization error for high-frequency two counter measurements:

$$\text{Err}_{\text{Quantization}} = \frac{\text{Actual Period}}{\text{Measurement Time}}$$

$$\text{Err}_{\text{Quantization}} = \frac{1}{(\text{Measurement Time} \times \text{Actual Frequency})}$$

Increasing the measurement time and the frequency input signals reduces the quantization error. Table 14-2 lists the quantization error for various measurement times and input signal frequencies. Notice that using higher input signal frequencies reduces quantization error.

Table 14-2. Quantization Error with High-Frequency Two Counter Method

Actual Frequency of Input Signal	Measurement Time	Quantization Error
10 kHz	1 ms	10.00%
100 kHz	1 ms	1.00%
1 MHz	1 ms	0.10%
10 MHz	1 ms	0.01%

Table 14-2. Quantization Error with High-Frequency Two Counter Method (Continued)

Actual Frequency of Input Signal	Measurement Time	Quantization Error
10 kHz	100 ms	0.10%
1 MHz	100 ms	0.001%
10 MHz	100 ms	0.0001%
10 kHz	1 s	0.010%
100 kHz	1 s	0.0010%
1 MHz	1 s	0.0001%
10 MHz	1 s	0.00001%

High-Frequency Two Counter Measurement Method Using NI-DAQmx

The block diagram in Figure 14-6 uses NI-DAQmx VIs to measure a signal with a frequency of approximately 10 MHz. The **Starting Edge** input is set to `Rising`, which means the counter begins taking the measurement when it encounters the first rising edge. The DAQmx Read VI returns the frequency in Hertz.

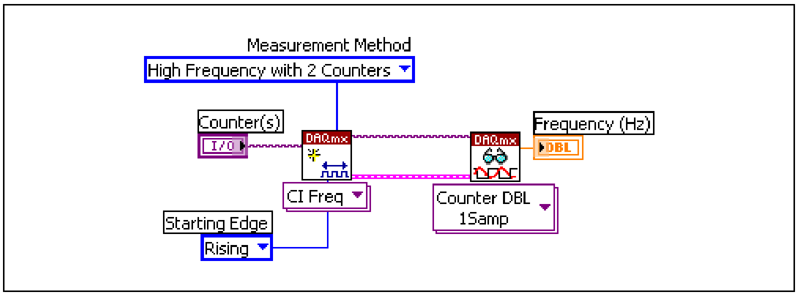


Figure 14-6. Using NI-DAQmx to Measure Frequency

Note Refer to the *NI-DAQmx Help* for more information about signal connections for two counter measurements.

Large-Range Two Counter Measurement Method

Use the large-range two counter measurement method if you measure the digital frequency or period of a signal with large frequency ranges. This method is useful when you measure a widely varying signal and want to increase accuracy throughout the entire range.

The hardware configuration is exactly the same as the high-frequency two counter measurement method. However, NI-DAQ uses the second counter to divide the input signal by the Divisor property. The Divisor property shifts the measurable frequency range upward and can cause the count register to roll over. The Divisor property scales the measured period and returns data according to the following equations:

$$\text{Period} = \frac{\text{Measured Period}}{\text{Divisor}}$$

$$\text{Frequency} = \text{Divisor} \times \text{Measured Frequency}$$

For example, if you use a 24-bit counter and the Counter Timebase Rate is 100 kHZ, the measurable frequency range is approximately 0.006 Hz to 50 kHz because

$$\text{Frequency} = \left(\frac{\text{Counter Timebase}}{\text{Count}}\right) \times \text{Divisor}$$

$$\text{Frequency} = \left(\frac{100\text{ kHz}}{2^{24}}\right) \times 1 = .006\text{ Hz and } \left(\frac{100\text{ kHz}}{2}\right) \times 1 = 50\text{ kHz}$$

However, with a divisor of 4, the measurable frequency range is 0.024 Hz to 200 kHz because

$$\text{Frequency} = \left(\frac{\text{Counter Timebase Rate}}{\text{Count}}\right) \times \text{Divisor}$$

$$\text{Frequency} = \left(\frac{100\text{ kHz}}{2^{24}}\right) \times 4 = .024\text{ Hz and } \left(\frac{100\text{ kHz}}{2}\right) \times 4 = 200\text{ kHz}$$

Quantization Error with Large-Range Two Counter Measurement Method

Use the following equations to calculate quantization error for large-range two counter measurements:

$$\mathrm{Err}_{\mathrm{Quantization}} = \frac{1}{(\mathrm{Divisor} \times \mathrm{Counter\ Timebase\ Rate} \times \mathrm{Actual\ Period} - 1)}$$

$$\mathrm{Err}_{\mathrm{Quantization}} = \frac{\mathrm{Actual\ Frequency}}{(\mathrm{Divisor} \times \mathrm{Counter\ Timebase\ Rate} - \mathrm{Actual\ Frequency})}$$

Increasing the divisor, increasing the counter timebase rate, or lowering the input signal frequency reduces the quantization error. Table 14-3 lists the quantization error for various divisors and input signal frequencies assuming a counter timebase rate of 20 MHz.

Table 14-3. Quantization Error with Large-Two Range Two Counter Method

Actual Frequency of Input Signal	Divisor	Quantization Error
1 kHz	4	0.00125%
10 kHz	4	0.0125%
100 kHz	4	0.125%
1MHz	4	1.25%
10 MHz	4	12.5%
1 kHz	10	0.0005%
10 kHZ	10	0.005%
100 kHz	10	0.05%
1 MHz	10	0.5%
10 MHz	10	5.0%
1 kHz	100	0.00005%
10 kHz	100	0.0005%
100 kHz	100	0.005%
1 MHz	100	0.05%
10 MHz	100	0.5%

Notice that the use of a divisor reduces the quantization error. Although the high-frequency two counter measurement method is more accurate at higher frequencies, the large-range two counter measurement method is more accurate throughout the range in a shorter amount of time. For example, if the input signal varies between 1 kHz and 1 MHz and you require a maximum quantization error of 2.0% at any signal range, you need a minimum measurement time of 50 ms using the high-frequency two counter measurement method. To gain the same accuracy using the large-range two counter method requires a maximum measurement time of 4 ms for any one measurement.

15 Generating Digital Pulses

This chapter describes how to generate a digital pulse using DAQ devices and FieldPoint modules.

Overview of Generating a Digital Pulse

Some measurement devices can generate a pulse signal from the device's counter/timer. The pulse is low (0 V) or high (5 V), as shown in Figure 15-1. Pulse generation uses a counter to output pulses.

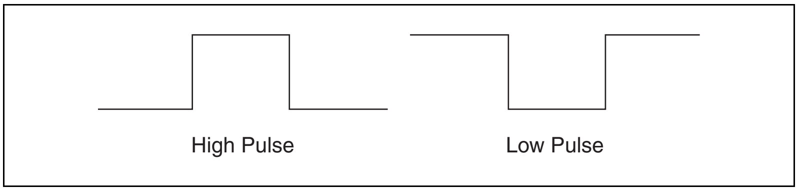


Figure 15-1. High Pulse and Low Pulse

You can use a pulse or pulse train, which is more than one pulse, as a clock signal, as a gate, or to trigger a measurement or a pulse generation. You can use a single pulse of known duration to determine an unknown signal frequency or to trigger an analog acquisition. You can use a pulse train of known frequency to determine an unknown pulse width.

Figure 15-2 shows the elements of a pulse, and Figure 15-3 shows the elements of a pulse train.

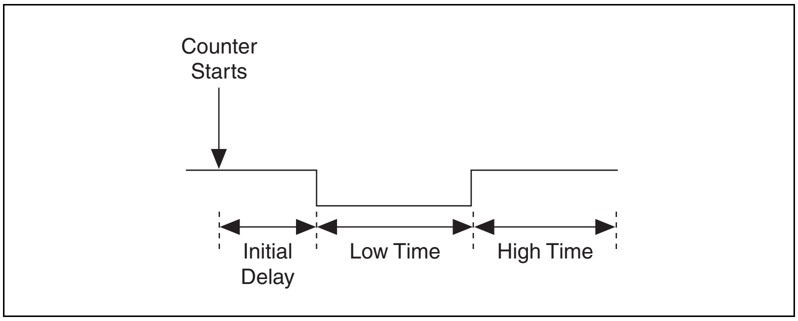


Figure 15-2. Elements of a Pulse

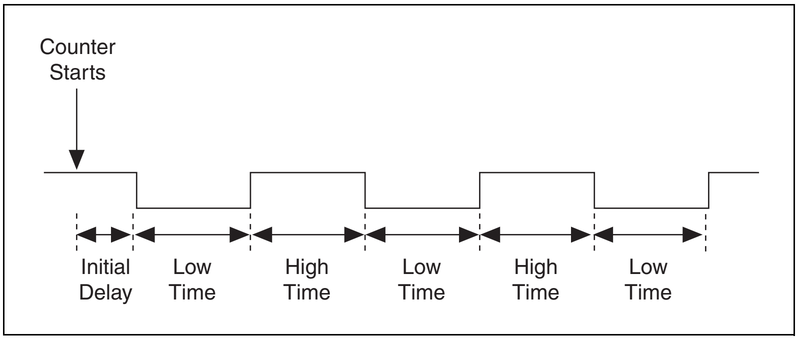


Figure 15-3. Elements of a Pulse Train

- The initial delay is the amount of time that the output remains at the idle state before generating the pulse.
- The high time is the amount of time the pulse is at a high level (5 V).
- The low time is the amount of time the pulse is at a low level (0 V).

The period of the pulse is the sum of the high time and the low time. The frequency is the reciprocal of the period (1/period).

The duty cycle, shown in Figure 15-4, is another characteristic of a pulse. Use the following equation to calculate the duty cycle of a pulse whose high time and low time are unequal:

$$\text{Duty Cycle} = \text{High Time}/\text{Pulse Period}$$

The duty cycle of a pulse is between 0 and 1 and is often expressed as a percentage. A pulse with a high time equal to the low time has a duty cycle of 0.5, or 50%. A duty cycle greater than 50% indicates that the high time is larger than the low time, and a duty cycle less than 50% indicates that the low time is greater than the high time.

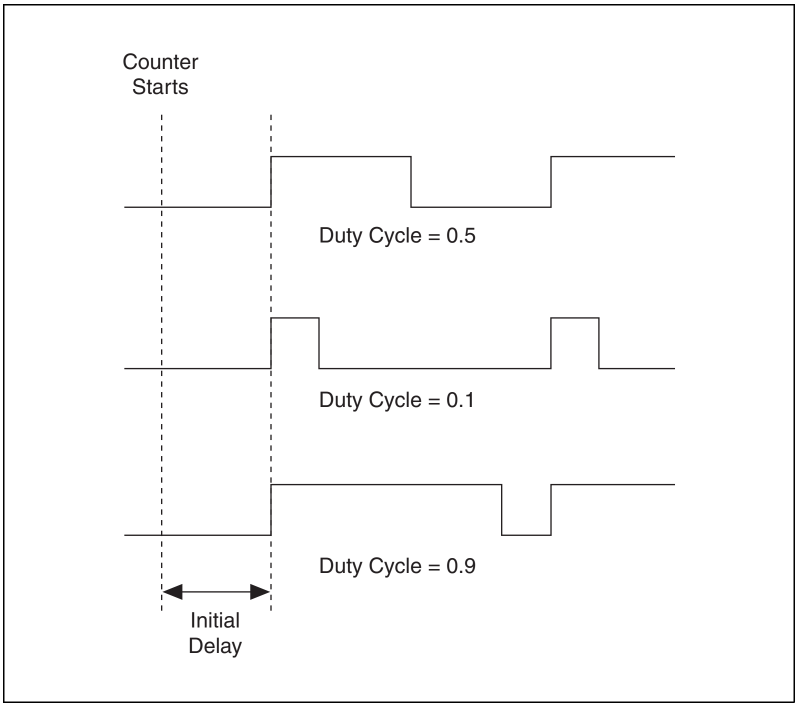


Figure 15-4. Duty Cycle of a Pulse

Before you generate a pulse, you also need to determine if you want to output the pulse or pulse train in terms of frequency, time, or number of ticks of the counter timebase. For frequency, you need to determine the duty cycle. For time, you specify the high time, or the amount of time that the pulse is high at 5 V, and the low time, or the amount of time that the pulse is low at 0 V. When you configure a pulse generation, the output appears at the counter output terminal.

The idle state controls the pulse generation polarity. When you set the idle state to low, the pulse generation starts low for the initial delay, transitions high for the high time, and transitions low for the low time, as shown in Figure 15-5. The high time and low time repeat for each pulse.

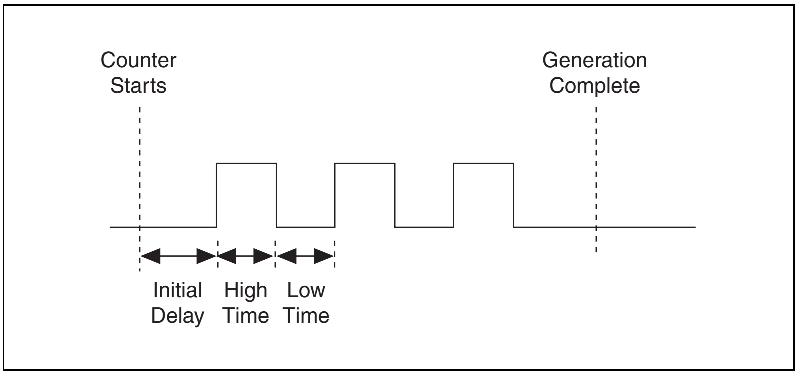


Figure 15-5. Low Idle State

When you set the idle state to high, the pulse generation starts high for the initial delay, transitions to low for the low time, and transitions high for the high time, as shown in Figure 15-6. In both cases, the output rests at the idle state when the pulse generation completes.

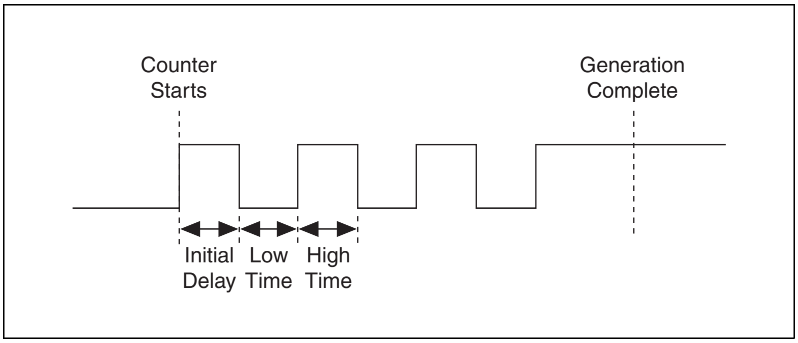


Figure 15-6. High Idle State

You can update the high time and low time of a continuous pulse train generation at any time, including while the application is running. This is useful for applications that require pulse width modulation, such as proportional integral derivative (PID) loop control applications.

Using NI-DAQmx VIs to Generate a Digital Pulse

The block diagram in Figure 15-7 uses NI-DAQmx VIs to generate a pulse train. The NI-DAQmx Create Channel VI defines the parameters of the pulse train. In Figure 15-7, the idle state is low, and the pulse has a frequency of 10 Hz with a 50% duty cycle, which means that the pulse starts low, transitions to high for 50 ms, and transitions to low for 50 ms. The DAQmx Timing VI sets up the counter to generate five pulses and stops. The DAQmx Start VI arms the counter and starts generating a pulse. The DAQmx Wait Until Done VI makes sure the application generates the pulses before the application finished executing. If you do not use the DAQmx Wait Until Done VI, the application might finish executing before you generate all five pulses.

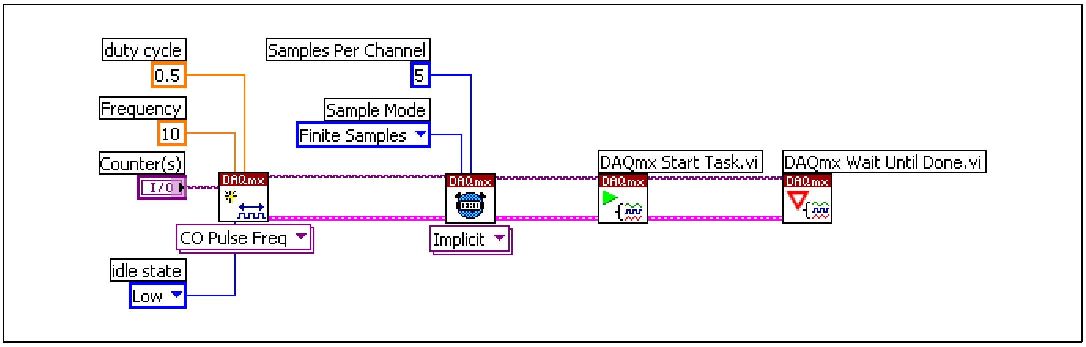


Figure 15-7. Using NI-DAQmx VIs to Generate Pulse

Using FieldPoint VIs to Generate a Digital Pulse

The block diagram in Figure 15-8 uses FieldPoint VIs to vary the parameters of a digital pulse. In this example, the FieldPoint I/O Point control is used with the cFP-PG-522 pulse generator module. This module continuously generates pulses, and you can vary the pulse on time, pulse off time, pulse mode, and resolution.

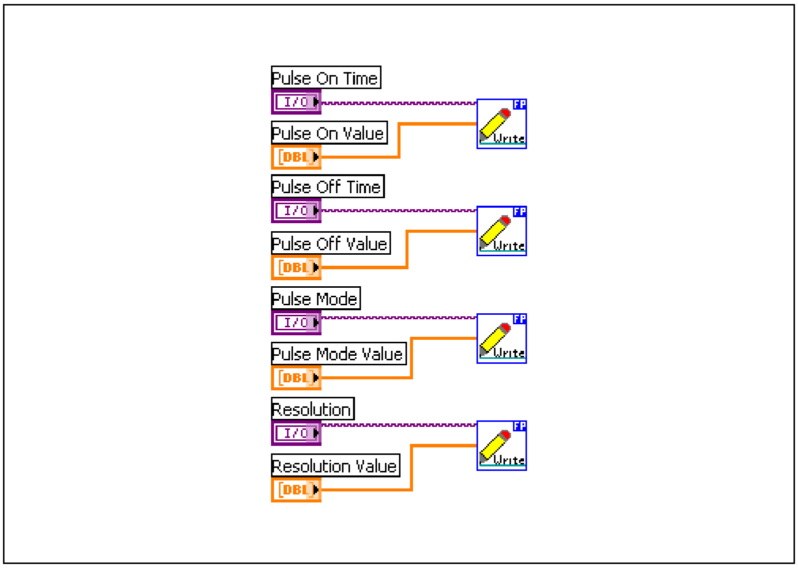


Figure 15-8. Using FieldPoint VIs to Generate Pulse

16

Using LabVIEW to Control Instruments

This chapter describes how to use instrument drivers and VISA to communicate with instruments.

Overview of Instrument Drivers

You control instruments by sending commands and data between the instrument and the PC. With LabVIEW, you can use an instrument driver or VISA to write customized VIs.

An instrument driver is a set of software routines that control a programmable instrument. Each routine corresponds to a programmatic operation, such as configuring, reading from, writing to, or triggering the instrument. Instrument drivers simplify instrument control and reduce test program development time by eliminating the need to learn the programming protocol for each instrument. The LabVIEW Instrument Driver Library contains instrument drivers for a variety of programmable instruments, including GPIB, VXI, and RS-232/422 instruments. Because instrument driver VIs contain high-level functions with intuitive front panels, you can quickly test and verify the remote capabilities of the instrument without the knowledge of device-specific syntax. You can create instrument control applications and systems by programmatically linking instrument driver VIs on the block diagram.

LabVIEW instrument drivers usually use VISA functions to communicate with instruments. VISA is the underlying protocol used when talking to instruments. You can use VISA for many different instrument types, such as GPIB, serial, PXI, and VXI. Once you learn how to communicate using VISA for one type of instrument, you do not have to learn a different way to communicate when you need to use another type of instrument. You do have to learn about the specific command set for the two instruments, but the method by which you send and receive the commands does not change.

Installing Instrument Drivers

You can download instrument drivers from the Instrument Driver Network at `ni.com/idnet`.

If an instrument driver for a particular instrument does not exist, try the following alternatives:

- Use a driver for a similar instrument. Often, similar instruments from the same manufacturer have similar, if not identical, command sets.
- Use NI-VISA to develop VIs that can communicate with the instrument. You can also use the Instrument I/O Assistant Express VI to communicate with an instrument without an instrument driver.
- Develop a complete, fully functional instrument driver. Refer to the Instrument Driver Network at `ni.com/idnet` for information about developing a National Instruments instrument driver.

Instrument Driver Directory

Install instrument drivers as a subdirectory of the `labview\instr.lib` directory. For example, the HP34401A instrument driver, included with LabVIEW, is installed in the `labview\instr.lib\hp34401a` directory.

The menu files and VI libraries that make up an instrument driver also are in this directory. The menu files allow you to view the instrument driver VIs on the **Functions** palette. The VI libraries contain the instrument driver VIs.

Organization of Instrument Drivers

Figure 16-1 shows the organization of a typical instrument driver.

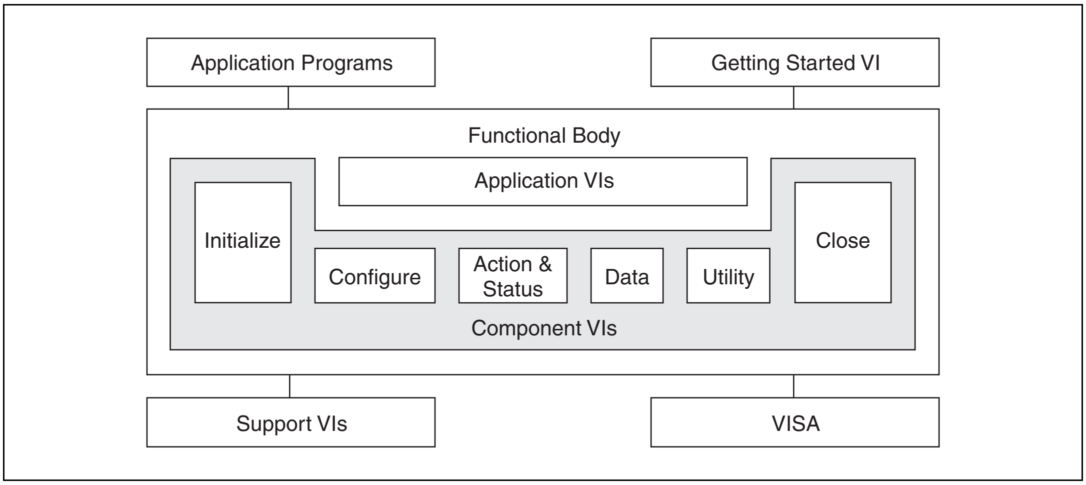


Figure 16-1. Instrument Driver Model

Use the Getting Started VI to verify communication with an instrument. The Getting Started VI consists of three subVIs: the Initialize VI, the Application VI, and the Close VI.

The Application VIs are high-level examples of grouping together low-level component functions to execute a typical programmatic instrument operation. For example, the Application VIs might include VIs to control the most commonly used instrument configurations and measurements. These VIs serve as examples to execute common operations such as configuring the instrument, triggering, and taking a measurement.

Because the Application VIs are standard VIs with icons and connector panes, you can call them from any high-level application when you want a single, measurement-oriented interface to the driver. For many users, the Application VIs are the only instrument driver VIs you need for instrument control. Refer to the HP34401A App. Example VI for an example of an Application VI.

The Initialize VI, the first instrument driver VI you call, establishes communication with the instrument. Additionally, it can perform any necessary actions to place the instrument in its default power on state or in another specific state. Generally, you need to call the Initialize VI only once at the beginning of an application.

The Configuration VIs are a collection of software routines that configure the instrument to perform the operation you want. Numerous Configuration VIs can exist, depending on the particular instrument. After you call these VIs, the instrument is ready to take measurements or to stimulate a system.

The Action VIs initiate or terminate test and measurement operations, such as arming the trigger system or generating a stimulus. Action VIs are different from Configuration VIs because they do not change the instrument settings but order the instrument to carry out an action based on its current configuration. The Status VIs obtain the current status of the instrument or the status of pending operations.

The Data VIs transfer data to or from the instrument. Examples include VIs for reading a measured value or waveform and VIs for downloading waveforms or digital patterns to a source instrument.

The Utility VIs perform a variety of operations that are auxiliary to the most often used instrument driver VIs. These VIs include the majority of the instrument driver template VIs, such as reset, self-test, revision, error query, and error message. The Utility VIs might also include other custom instrument driver VIs that perform operations such as calibration or storage and recall of setups.

The Close VI terminates the software connection to the instrument and frees system resources. Generally, you need to call the Close VI only once at the end of an application or when you finish communication with an instrument. Make sure that for each successful call to the Initialize VI, you use a matching Close VI to avoid maintaining unnecessary memory resources.

Note Application VIs do not call the Initialize VI and Close VI. To run an application VI, you first must run the Initialize VI. The Getting Started VI calls the Initialize VI and Close VI.

Types of Instrument Drivers

Three common types of instrument drivers exist to control instruments in LabVIEW. The difference is not in how you use them but in how you implement them. The three types of instrument drivers are:

- LabVIEW Plug and Play drivers
- IVI drivers
- Contributed instrument drivers

LabVIEW Plug and Play Drivers

A LabVIEW Plug and Play instrument driver is a set of VIs that controls and communicates with a programmable instrument. Each VI corresponds to a programmatic operation, such as configuring, reading from, writing to, or triggering an instrument. LabVIEW Plug and Play instrument drivers include error handling, front panels, block diagrams, icons, and online help. Because LabVIEW Plug and Play drivers maintain a common architecture and interface, you can quickly connect to and communicate with an instrument with very little or no code development.

IVI Drivers

IVI drivers are more sophisticated drivers that allow for simulation and instrument interchangeability. You do not have to rewrite an application to use it with a similar instrument type. For example, you can write a VI that works with several different brands of oscilloscopes, even if those oscilloscopes use different bus connections. To achieve interchangeability, the IVI Foundation, which develops IVI standards, defines specifications for the following instrument classes: DMM, oscilloscope, arbitrary waveform/function generator, DC power supply, switch, power meter, spectrum analyzer, and RF signal generator.

Use National Instruments IVI drivers for the following additional benefits:

- Instrument state caching for improved performance
- Multithread safety
- Instrument attribute access

Contributed Instrument Drivers

Contributed instrument drivers are included "as is" and typically solve a specific application instead of featuring a fully functional instrument driver. Contributed instrument drivers are not supported by NI or other third parties.

VISA in LabVIEW

VISA is a standard I/O API for instrumentation programming. VISA can control GPIB, serial, Ethernet, PXI, or VXI instruments, making the appropriate driver calls depending on the type of instrument you use.

Message-Based Communication versus Register-Based Communication

GPIB, serial, Ethernet, and some VXI instruments use message-based communication. You program message-based instruments with high-level ASCII character strings. The instrument has a local processor that parses the command strings and sets the appropriate register bits to perform the operations you want. The Standard Commands for Programmable Instruments (SCPI) standardizes the ASCII command strings used to program any instrument. Similar instruments use similar commands. Instead of learning different command messages for each type of instrument from each manufacturer, you need to learn only one command set. The most common message-based functions are VISA Read, VISA Write, VISA Assert Trigger, VISA Clear, and VISA Read STB.

PXI and many VXI instruments use register-based communication. You program register-based instruments at a low level using binary information that you write directly to the instrument control registers. Speed is the advantage of this type of communication because the instrument no longer needs to parse the command strings and convert the information to register-level programming. Register-based instruments literally communicate at the level of direct hardware manipulation. The most common register-based functions are VISA In, VISA Out, VISA Move In, and VISA Move Out.

Refer to the *Writing VISA Applications* section of this chapter for more information about using VISA.

Verifying Communication with an Instrument

You can use several features to verify communication with an instrument and to test a typical programmatic instrument operation. The following are common reasons why communication with an instrument fails:

- The instrument is not properly connected or configured.
- NI-VISA is not installed. If you did not install NI-VISA when you installed LabVIEW, you must install it before you use LabVIEW features to verify communication with an instrument.
- The instrument address was incorrect. The Getting Started VI requires the correct address for an instrument. If you are not certain of the instrument address, use MAX or the VISA Find Resource function to confirm the instrument address.
- The instrument driver does not support the exact model of instrument you are using.

Instrument I/O Assistant

(Windows) Use the Instrument I/O Assistant utility to communicate with a GPIB, serial, or Ethernet instrument and graphically parse the response. Instrument I/O Assistant organizes instrument communication into ordered steps. You can use Instrument I/O Assistant to send a query to an instrument to verify communication with that instrument.

Place the Instrument I/O Assistant Express VI on the block diagram to access Instrument I/O Assistant.

To validate communication, use a Query and Parse step and send an identification command to the instrument (`*IDN?` for most instruments). If the instrument responds, you have established communication with the instrument. If the instrument returns a timeout error, ensure that the instrument is properly connected to the computer, powered on, and properly configured. Refer to the instrument documentation for more information about connecting and configuring the instrument.

Verifying VISA Communication

If no VISA VIs or instrument drivers appear to work in LabVIEW, use the VISA Find Resource function. This function runs without any other VISA VIs or functions on the block diagram. If the VISA Find Resource function returns errors, you most likely have the wrong version of VISA installed, or VISA is not installed correctly. If the VISA Find Resource function executes correctly, which indicates that LabVIEW is working correctly with the VISA driver, you must then identify what sequence of VIs produces the error in LabVIEW.

If a simple sequence of events produces the error, try the same sequence interactively with the VISA Interactive Control (VISAIC) utility. **(Windows)** Select **Start»National Instruments»VISA»VISA Interactive Control** to launch the VISAIC utility. You also can select **Tools»VISA»VISA Interactive Control** in MAX to launch the VISAIC utility.

If the interactive utility works successfully but the same sequence in LabVIEW does not, LabVIEW might have a problem interacting with the VISA driver. If the sequence exhibits the same problem interactively in VISAIC, a problem might exist with one of the drivers VISA calls.

Getting Started VI

The Getting Started VI can verify communication with an instrument and test a typical programmatic instrument operation. Review and set each of the controls in the Getting Started VI. With the exception of the address field, the defaults for most controls are usually sufficient for the first time you run this VI. **(Windows)** Refer to MAX if you do not know the address of the instrument.

After you run the VI, verify that it returned the kind of data you expect and that it did not report an error in the error cluster.

Customizing the Getting Started VI for Measurements

After you use the Getting Started VI to verify basic communication, you can edit the VI for the instrument control needs. Before you edit the VI, save a copy of it by selecting **File»Save As**. Select **Operate»Make Current Values Default** to change the default values on the front panel. Block diagram changes might include changing the constants wired to the Application VI or other subVIs.

Common Instrument Driver VIs Inputs and Outputs

Just as all instrument drivers share a common set of functions, they also share common inputs and outputs.

Resource Name/Instrument Descriptor

When you initialize an instrument with an Initialize Instrument Driver VI, you need to know the resource name or instrument descriptor. The resource name is the VISA alias or IVI logical name. The instrument descriptor is the exact name and location of a resource and has the following format:

```
Interface Type[board index]::Address::INSTR
```

For example, `GPIB0::2::INSTR` is the instrument descriptor when you use the first GPIB board to communicate with an instrument at device address `2`.

(Windows) Use MAX to determine the available resources and instrument addresses. Refer to the *Configuring VISA Devices and IVI Logical Names* section of Chapter 3, *Configuring Measurement Hardware*, for more information about using VISA aliases.

Error In/Error Out Clusters

Error handling with instrument driver VIs is similar to error handling with other I/O VIs in LabVIEW. Each instrument driver VI contains an **error in** input and an **error out** output for passing error clusters from one VI to another. The error cluster contains a Boolean flag that indicates if an error occurred, an error code number, and a string that contains the location of the VI where the error occurred.

Writing VISA Applications

For most simple instrument applications, you need only two VISA functions: VISA Write and VISA Read, as shown in Figure 16-2.

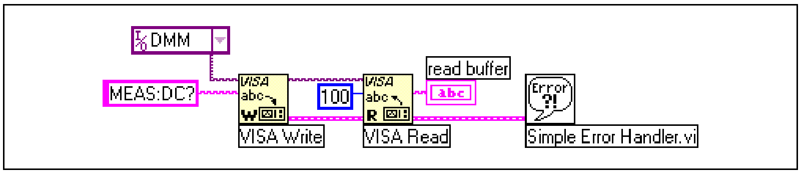


Figure 16-2. VISA Example

The VISA resource name constant specifies which instrument to use. The VISA Write function determines if a reference is already established with the specified instrument. If a reference does not exist, a reference automatically opens, and the VISA Write VI sends the string `MEAS:DC?` to the instrument.

You can wire the **VISA resource name** output of the VISA Write function to the VISA Read function to specify the instrument to read. You can process and display the returned output from the VISA Read function as necessary for the measurement. The Simple Error Handler VI processes any errors that occurred with the VISA functions.

Using VISA Properties

VISA resources have a variety of properties (attributes) with values you can read or set. You can use the Property Node to read or set the values of VISA properties.

After you place a Property Node on the block diagram, wire a VISA Session to the **reference** input of the Property Node. Once you wire a session to the **Session** input of the Property Node, LabVIEW sets the VISA Class to the class associated with that session.

To optionally change the VISA class, right-click the Property Node and select **Select Class»VISA»I/O Session** from the shortcut menu. The default is `INSTR` class, which encompasses all VISA properties. The classes limit the properties displayed in the shortcut menu to those related to the selected class instead of to all the VISA properties.

There are two basic types of VISA properties: global properties and local properties. Global properties are specific to a resource, and local properties are specific to a session. A global property applies to all the sessions that are open to that resource. A local property is a property that can be different for individual sessions to a specific resource.

Refer to the **Context Help** window for brief descriptions of individual properties. Refer to the *LabVIEW Help*, available by selecting **Help»VI, Function, & How-To Help**, for more information about using VISA properties.

Using VISA Events

An event is a means of VISA communication between a resource and its applications, in which the resource notifies the application that some condition has occurred that requires action by the application.

Handling GPIB SRQ Events Example

The block diagram in Figure 16-3 uses VISA to handle GPIB Service Request (SRQ) events.

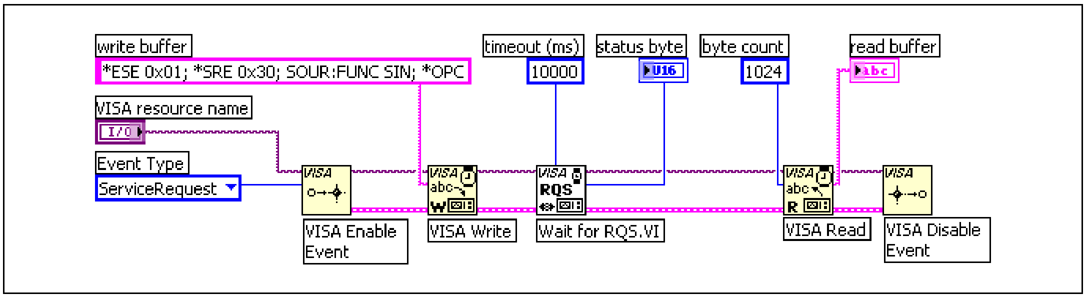


Figure 16-3. Handling GPIB Events Example

The VI in Figure 16-3 enables service request events and writes a command string to the instrument. The VI expects the instrument to respond with an SRQ when it has processed the string. The Wait for RQS VI waits for up to 10 seconds for the SRQ event to occur. After the SRQ occurs, the VISA Read function reads the instrument status byte. The VI must read the status byte after GPIB SRQ events occur or it might not receive later SRQ events properly. The VI reads the response from the instrument and displays the response.

Using Advanced VISA VIs

Use the VIs in the **Advanced VISA** palette to build advanced VISA VIs. Refer to `\examples\instr\visa.llb` for examples of using advanced VISA VIs.

Data and String Manipulation Techniques

Communicating with instruments involves sending data to instruments and retrieving data from instruments. You rarely need to format the data sent to or retrieved from the instrument when using instrument drivers because the instrument driver handles the formatting for you. However, you might need to format the data when writing VIs to communicate with instruments.

When you communicate with a message-based instrument, you must format and build the correct command strings for the instrument to perform the appropriate operation or return a response.

No standards exist for register-based instrument communication. Each device operates differently, and the instrument documentation is the best resource for learning how to program the device.

Typically, a command string, or query, is a combination of text and numeric values. Some instruments require text-only command strings, requiring you to convert the numeric values to text and append them to the command string. Similarly, to use the data an instrument returns in LabVIEW, you must convert the data to a format a VI, function, or indicator can accept.

Using Instrument I/O Assistant for Data Manipulation

(Windows) You can use Instrument I/O Assistant to send queries to an instrument and to format the data an instrument returns. Place the Instrument I/O Assistant Express VI on the block diagram to access Instrument I/O Assistant.

Formatting Commands into Strings

Use the Format Into String function to build command strings to send to an instrument. You can use the Format into String function to take an initial string and append other strings or numeric data types to it.

The block diagram in Figure 16-4 formats text and data into a command string.

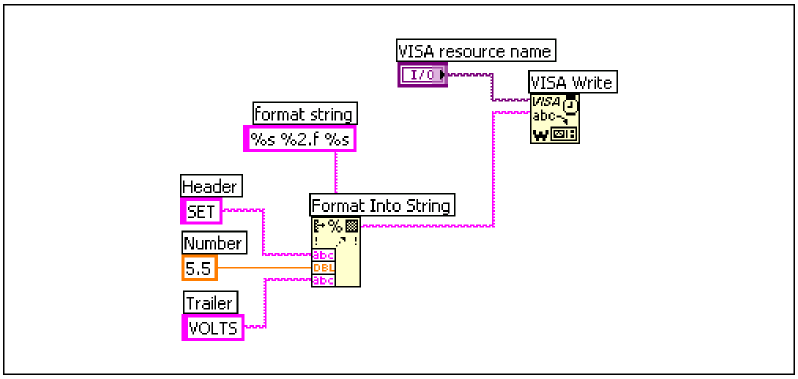


Figure 16-4. Formatting Commands into Strings

The Format Into String function creates a string, `SET 5.50 VOLTS`, which the VISA Write VI accepts as a command to set the instrument to generate 5.5 volts. `SET` is the header information, and `VOLTS` is the trailer information.

Formatting Data Retrieved from an Instrument

Just as the commands strings you send to an instrument contain headers and trailers, most instruments return data with headers and/or trailers. The header usually contains information such as the number of data points returned or the instrument settings. Trailer information often contains units or other instrument settings at the end of the data string. The instrument documentation should describe what header and trailer information to expect from each data transfer. You must first remove the header and trailer information before you can display or analyze the returned data in LabVIEW.

The block diagram in Figure 16-5 sends a command to return a voltage reading from an instrument. The VISA Read function returns the reading as a string. In this example, suppose the instrument returns a response such as `VOLTS:DC 12.3456789 volts`. The two String Subset functions parse the header and trailer information in the string and display them in string indicators. The Scan From String function parses the data from the string. The constant value of `9` wired to the **offset** input of the Scan from StringF function removes the header information from the string. Different instruments respond to commands differently. Figure 16-5 is one example.

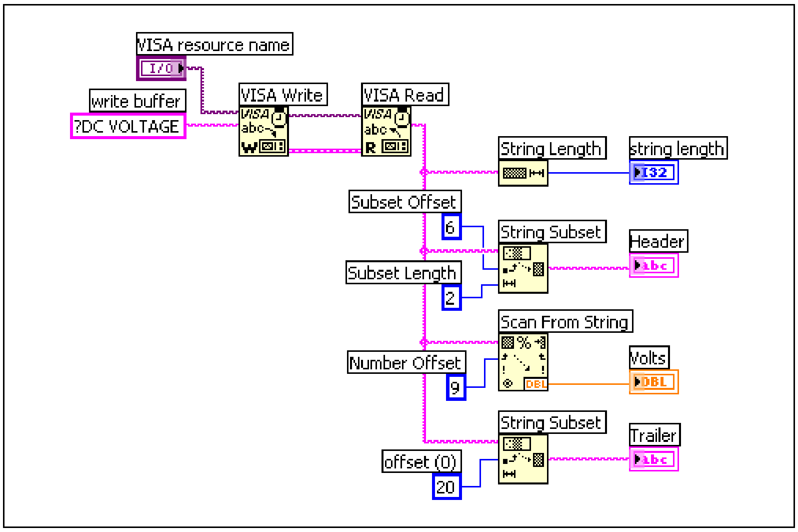


Figure 16-5. Formatting Data Retrieved from an Instrument

Waveform Transfers

Instruments also can return data in other formats, such as ASCII, 1-byte binary, and 2-byte binary formats. The instrument describes available formats and how to convert each one to usable data.

ASCII Waveforms

If an instrument returns data in ASCII format, you can view the data as a character string. However, if you need to manipulate the data in numeric format or if you need to graph the data, you must convert the string data to numeric data. As an example, consider a waveform composed of 1,024 points and in which each point has a value between 0 and 255. Using ASCII encoding, you need a maximum of 4 bytes to represent each data value (a maximum of 3 bytes for the value and 1 byte for the separator, such as a comma). You need a maximum of 4,096 bytes (4 bytes × 1,024) plus any header and trailer bytes to represent the waveform as an ASCII string.

The block diagram in Figure 16-6 uses the Extract Numbers VI, which is an example VI, to convert the ASCII string the VISA Read function returns to a numeric array. The Extract Numbers VI finds all numbers in the given ASCII string and returns a single-precision array of numbers. The Extract Numbers VI ignores any characters at the beginning of the ASCII string, so you do not need to remove header information when you use the Extract Numbers VI.

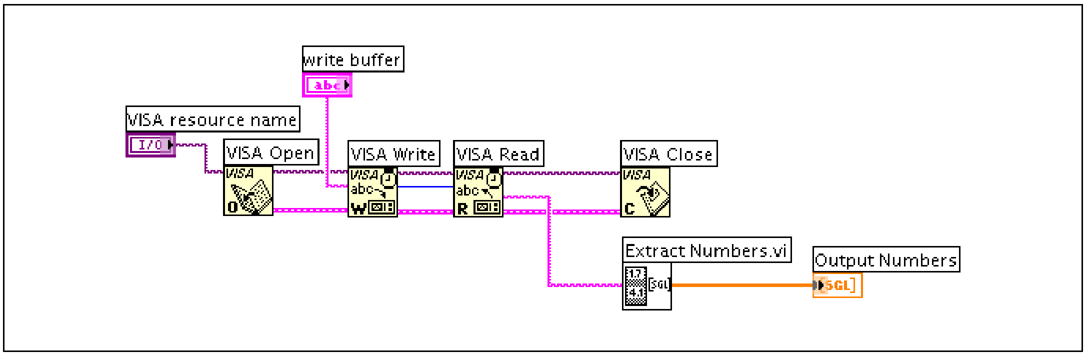


Figure 16-6. ASCII Waveforms

1-Byte Binary Waveforms

Some instruments do not have the option of sending data in ASCII format or send all waveform data in binary format. Because no standard exists for binary format, refer to the instrument documentation to determine exactly how the instrument stores the data values. One common binary format is 1-byte binary. With this type of data encoding, the instrument converts each piece of data to an 8-bit binary value before sending it.

When you read 1-byte binary data from the bus, the instrument returns the data as a character string. However, the characters do not appear to correspond to the expected data. The binary numbers are interpreted as ASCII character values, and displays the corresponding characters. For example, if you send the value of 65 as one data value, you read the character A from the bus. For a value of 13, no printable ASCII character exists because 13 corresponds to an invisible carriage return character.

You can display these invisible characters in a string indicator by right-clicking the indicator and selecting **'\'Codes Display** from the shortcut menu. The carriage return character is `\r` in the string indicator.

To use the numeric data in an ASCII string with the Analysis VI or to display the numeric data in a graph or chart, you must convert the binary string to a numeric array. If the instrument sends a binary string that contains 1,024 1-byte binary encoded values, the waveform requires only 1,024 bytes plus any header information. Using binary encoding, you need only 1 byte to represent each data value, assuming each value is an unsigned 8-bit integer.

To convert the binary string to a numeric array, you first use the String Subset function to remove all header information and trailer information. Then you can use the String To Byte Array function as shown in Figure 16-7 to convert the remaining data string to an array of integers.

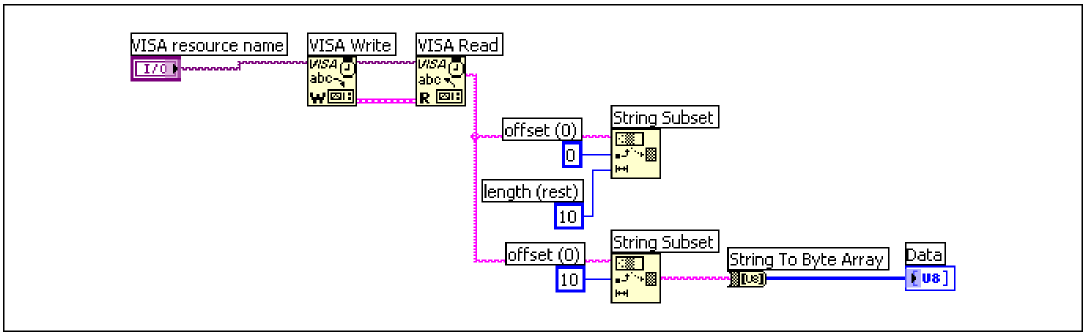


Figure 16-7. 1-Byte Binary Waveforms

Note When acquiring binary data, extract the data using the data size rather than searching for the first character of the trailer information because that character might be part of the binary values.

2-Byte Binary Waveforms

When data is in 2-byte binary format, it is binary encoded and sent as ASCII characters similar to the 1-byte binary format. However, 16 bits of data, or two ASCII characters, represent each data value. Although this format uses twice as much space as the 1-byte binary format, it is more efficiently packed than ASCII-formatted data.

As an example, consider an oscilloscope that transfers waveform data in binary notation. For this example, assume the waveform consists of 1,024 data points where each value is a 2-byte signed integer. Therefore, the entire waveform requires 2,048 bytes plus a 5-byte header and a 2-byte trailer. After you remove the 5-byte header, use the Type Cast function to convert the waveform string to an array of 16-bit integers.

Byte Order

It is important to know the order of the bytes you receive when data is transferred in 2-byte binary format. The 2-byte combination `qH` has the corresponding integer value of 29,000, but the opposite byte order of `Hq` has the corresponding integer value of 18,545.

Note You can configure the byte order for some instruments, and some instruments have a fixed byte order. Refer to the instrument documentation for specific instrument byte information.

If you receive the high byte first, you must reverse the order of the bytes before you convert them to an integer value.

A

Types of Instruments

When you use a PC to automate a test system, you are not limited to the type of instrument you can control. You can mix and match instruments from various categories, such as GPIB, serial, PXI, and modular instruments.

When you use PCs to control instruments, you need to understand the following about the instrument:

- The type of connector (pinouts) on the instrument
- The kind of cable needed (null-modem, number of pins, male/female)
- The electrical properties involved (signal levels, grounding, cable length restrictions)
- The communication protocols to use (ASCII commands, binary commands, data format)
- The kind of software drivers available

This appendix outlines the most common categories of instruments. Additional types of instruments that are not outlined in this appendix include image acquisition, motion control, Ethernet, CAMAC, parallel port, CAN, FieldBus, and other devices.

GPIB Communications

GPIB instruments offer test and manufacturing engineers the widest selection of vendors and instruments for general-purpose to specialized vertical market test applications. GPIB instruments are often used as stand-alone benchtop instruments where measurements are taken by hand. You can automate these measurements by using a PC to control the GPIB instruments.

Controllers, Talkers, and Listeners

To determine which device has active control of the bus, the GPIB protocol categorizes devices as controllers, talkers, or listeners to determine which device has active control of the bus. Each device has a unique GPIB primary address between 0 and 30. The Controller defines the communication links, responds to devices that request service, sends GPIB commands, and passes/receives control of the bus. Controllers instruct Talkers to talk and to place data on the GPIB. You can address only one device at a time to talk. The Controller addresses the Listener to listen and to read data from the GPIB. You can address several devices to listen.

Hardware Specifications

The GPIB is a digital, 24-conductor parallel bus. It consists of eight data lines (DIO 1–8), five bus management lines (EOI, IFC, SRQ, ATN, REN), three handshake lines (DAV, NRFD, NDAC), and eight ground lines. The GPIB uses an eight-bit parallel, byte-serial, asynchronous data transfer scheme, which means that whole bytes are sequentially handshaked across the bus at a speed that the slowest participant in the transfer determines. Because the unit of data on the GPIB is a byte (eight bits), the messages transferred are frequently encoded as ASCII character strings.

The following additional electrical specifications allow data to be transferred across the GPIB at the maximum rate of 1 Mbyte/s because the GPIB is a transmission line system:

- A maximum separation of 4 m between any two devices and an average separation of 2 m over the entire bus.
- A maximum cable length of 20 m.
- A maximum of 15 devices connected to each bus with at least two-thirds of the devices powered on.

If you exceed any of these limits, you can use additional hardware to extend the bus cable lengths or to expand the number of devices allowed.

You can obtain faster data rates with HS488 devices and controllers. HS488 is an extension to GPIB that most NI controllers support.

Serial Port Communication

Serial communication transmits data between a computer and a peripheral device, such as a programmable instrument or another computer. Serial communication uses a transmitter to send data one bit at a time over a single communication line to a receiver. Use this method when data transfer rates are low or you must transfer data over long distances. Most computers have one or more serial ports, so you do not need any extra hardware other than a cable to connect the instrument to the computer or to connect two computers to each other.

You must specify four parameters for serial communication: the baud rate of the transmission, the number of data bits that encode a character, the sense of the optional parity bit, and the number of stop bits. A character frame packages each transmitted character as a single start bit followed by the data bits.

Baud rate is a measure of how fast data moves between instruments that use serial communication.

Data bits are transmitted upside down and backwards, which means that inverted logic is used and the order of transmission is from least-significant bit (LSB) to most-significant bit (MSB). To interpret the data bits in a character frame, you must read from right to left and read 1 for negative voltage and 0 for positive voltage.

An optional parity bit follows the data bits in the character frame. The parity bit, if present, also follows inverted logic. This bit is included as a means of error checking. You specify ahead of time for the parity of the transmission to be even or odd. If you choose for the parity to be odd, the parity bit is set in such a way so the number of 1s add up to make an odd number among the data bits and the parity bit.

The last part of a character frame consists of 1, 1.5, or 2 stop bits that are always represented by a negative voltage. If no further characters are transmitted, the line stays in the negative (MARK) condition. The transmission of the next character frame, if any, begins with a start bit of positive (SPACE) voltage.

Rate of Data Transfer

You can calculate the maximum transmission rate in characters per second for a given communication setting by dividing the baud rate by the bits per character frame.

Serial Hardware Overview

The following examples are the most common recommended standards of serial port communication:

- RS-232 (ANSI/EIA-232 Standard) is used for many purposes, such as connecting a mouse, printer, or modem. It also is used with industrial instrumentation. Because of improvements in line drivers and cables, applications often increase the performance of RS-232 beyond the distance and speed in the standards list. RS-232 is limited to point-to-point connections between PC serial ports and devices.

- RS-422 (AIA RS-422A Standard) uses a differential electrical signal as opposed to the unbalanced (single-ended) signals referenced to ground with RS-232. Differential transmission, which uses two lines each to transmit and receive signals, results in greater noise immunity and longer transmission distances as compared to RS-232.
- RS-485 (EIA-485 Standard) is a variation of RS-422 that allows you to connect up to 32 devices to a single port and define the necessary electrical characteristics to ensure adequate signal voltages under maximum load. With this enhanced multidrop capability, you can create networks of devices connected to a single RS-485 serial port. The noise immunity and multidrop capability make RS-485 an attractive choice in industrial applications that require many distributed devices networked to a PC or other controller for data collection and other operations.

Connecting a Serial Device

If you have a serial device in a system, you first must obtain the pinout for that device and make sure you have the correct cable to connect the serial device to the computer. Determine if the device is data communications equipment (DCE) or data terminal equipment (DTE) and what settings it uses to communicate—baud rate, data bits, stop bits, parity, or handshaking (flow control).

PXI Modular Instrumentation

A modular instrumentation system based on PXI delivers a PC-based, high-performance measurement system.

PXI is completely compatible with CompactPCI and incorporates advanced timing and triggering features. PXI fills the gap between low-cost desktop PC solutions and high-end VXI and GPIB solutions by combining the industry standards of Windows, PCI, and CompactPCI.

You design a PXI system by selecting a controller (an embedded Pentium class or higher computer and peripherals), the chassis, and the modules. PXI modules can be anything from analog-to-digital, digital-to-analog, digital I/O, and multifunction input/output boards to image acquisition, motion control, and instruments like oscilloscopes, multimeters, serial data analyzers, and other custom instruments.

Modular Instruments

Modular instruments are made for several different platforms including PCMCIA (laptops), PCI (desktop computers), and PXI.

Modular instruments are an example of virtual instruments that consist of a PC-based instrument module, a computer, and application software.

The modular instrument uses the expandable memory, display options, Internet connectivity, and processor of the PC. You can measure voltage, current, and resistance using a modular instrument, and you can expand the capabilities of a virtual instrument through application software. You can create a data logger and automatically analyze the data you acquire. You also can generate reports. While you are making measurements, you can analyze and present information to make decisions immediately.

With application software, you can customize the capabilities of a virtual instrument to solve multiple test challenges. You also can upgrade the performance of a measurement system with PC technology that offers a more economical instrumentation solution than purchasing a single-function, stand-alone instrument.

B Technical Support and Professional Services

Visit the following sections of the National Instruments Web site at `ni.com` for technical support and professional services:

- **Support**—Online technical support resources include the following:
  - **Self-Help Resources**—For immediate answers and solutions, visit our extensive library of technical support resources available in English, Japanese, and Spanish at `ni.com/support`. These resources are available for most products at no cost to registered users and include software drivers and updates, a KnowledgeBase, product manuals, step-by-step troubleshooting wizards, conformity documentation, example code, tutorials and application notes, instrument drivers, discussion forums, a measurement glossary, and so on.
  - **Assisted Support Options**—Contact NI engineers and other measurement and automation professionals by visiting `ni.com/support`. Our online system helps you define your question and connects you to the experts by phone, discussion forum, or email.
- **Training**—Visit `ni.com/custed` for self-paced tutorials, videos, and interactive CDs. You also can register for instructor-led, hands-on courses at locations around the world.
- **System Integration**—If you have time constraints, limited in-house technical resources, or other project challenges, NI Alliance Program members can help. To learn more, call your local NI office or visit `ni.com/alliance`.

If you searched `ni.com` and could not find the answers you need, contact your local office or NI corporate headquarters. Phone numbers for our worldwide offices are listed at the front of this manual. You also can visit the Worldwide Offices section of `ni.com/niglobal` to access the branch office Web sites, which provide up-to-date contact information, support phone numbers, email addresses, and current events.

Glossary

Symbol	Prefix	Value
n	nano	10^{-9}
μ	micro	10^{-6}
m	milli	10^{-3}
k	kilo	10^{3}
M	mega	10^{6}

A

A/D	Analog-to-digital; analog/digital.
ADC	Analog-to-digital converter. An electronic device, often an integrated circuit, that converts an analog voltage to a digital number.
AI	Analog input.
AIGND	Analog input ground pin on a DAQ device.
alias	A false lower frequency component that appears in sampled data acquired at too low a sampling rate compared to the Nyquist frequency.
amplification	To strengthen a signal. Often used to improve the accuracy of low-amplitude signals.
analog trigger	Trigger that occurs at a user-selected level and slope on an incoming analog signal. You can set triggering to occur at a specified voltage on either an increasing or a decreasing signal (positive or negative slope).
ANSI	American National Standards Institute.
AO	Analog output.
Application Programming Interface (API)	A library of functions, classes or VIs, attributes, and properties for creating applications for a device.

B

Butterworth filter	A filter with low ripple.

C

channel	1. Physical—a terminal or pin at which you can measure or generate an analog or digital signal. A single physical channel can include more than one terminal, as in the case of a differential analog input channel or a digital port of eight lines. The name used for a counter physical channel is an exception because that physical channel name is not the name of the terminal where the counter measures or generates the digital signal. 2. Virtual—a collection of property settings that can include a name, a physical channel, input terminal connections, the type of measurement or generation, and scaling information. You can define NI-DAQmx virtual channels outside a task (global) or inside a task (local). Configuring virtual channels is optional in Traditional NI-DAQ and earlier versions, but is integral to every measurement you take in NI-DAQmx. In Traditional NI-DAQ, you configure virtual channels in MAX. In NI-DAQmx, you can configure channels as part of a task or separately.
clock	Hardware component that controls timing for reading from or writing to groups.
cold-junction compensation	1. A method of compensating for inaccuracies in thermocouple circuits. 2. An artificial reference level that compensates for ambient temperature variations in thermocouple measurement circuits. IC temperature sensors are linear and their output is expressed as mV/°C. A 10 mV/°C sensor, for example, outputs 250 mV at 25 °C. Thermistor outputs, however, are nonlinear. Therefore, thermistor output is specified as the voltage range over a defined temperature range (x volts at 50 °C to y volts at 0 °C).
common-mode voltage	Any voltage present at the instrumentation amplifier inputs with respect to amplifier ground.
configuration utility	Refers to Measurement & Automation Explorer on Windows and to the NI-DAQ Configuration Utility on Macintosh.
coupling	Manner in which a signal connects from one location to another.

CMRR	Common-mode rejection ratio—measure of the capability of an instrument to reject a signal that is common to both input leads. For instance, if you measure a thermocouple in a noisy environment, the noise from the environment appears on both input leads. Therefore, this noise is a common-mode voltage signal that is rejected by an amount equal to the CMRR of the instrument. The CMRR is defined by the following equation: CMRR = 20 log(Differential Gain/Common Mode Gain) The ratio is important because it indicates how much of the common mode signal appears in your measurement. The value of the CMRR depends on signal frequency as well and must be specified as a function of frequency. An equivalent equation to represent CMRR is as follows: 20 Log(Measured Common Voltage/Applied Common Voltage)
counter	A circuit that counts pulses or clock cycles (timing). Counters and timers usually have from 16 bits to 48 bits (sometimes more) counting capability. The total number of counts possible equals 2^N, where N is the number of bits in the counter. When the events counted are the number of clock cycles from a clock source, the amount of time can be computed if the clock frequency is known.
curve fitting	A technique for extracting a set of curve parameters or coefficients from the data set to obtain a functional description of the data set.

D

D/A	Digital-to-analog.
DAC	Digital-to-analog converter. An electronic device, often an integrated circuit, that converts a digital number to a corresponding analog voltage or current.
DAQ	*See* data acquisition.
DAQ Assistant	A graphical interface for configuring measurement tasks, channels, and scales.
DAQ device	A device that acquires or generates data and can contain multiple channels and conversion devices. DAQ devices include plug-in drivers, PCMCIA cards, and DAQPad devices, which connect to a computer USB or 1394 (FireWire) port. SCXI modules are considered DAQ devices.

data acquisition	1. Acquiring and measuring analog or digital electrical signals from sensors, transducers, and test probes or fixtures. 2. Generating analog or digital electrical signals.
dB	Decibels.
device	An instrument or controller you can access as a single entity that controls or monitors real-world I/O points. A device often is connected to a host computer through some type of communication network. *See also* DAQ device *and* measurement device.
device number	Slot number or board ID number assigned to the device when you configured it.
DFT	*See* Discrete Fourier Transform.
differential measurement system	A way to configure your device to read signals in which you do not need to connect either input to a fixed reference, such as a building ground.
digital trigger	TTL signal that you can use to start or stop a buffered data acquisition operation, such as buffered analog input or buffered analog output.
Discrete Fourier Transform	A digital technique for computing the Fourier Transform.
DMA	Direct Memory Access. A method by which you can transfer data to computer memory from a device or memory on the bus, or from computer memory to a device, while the processor does something else. DMA is the fastest method of transferring large amounts of data to or from computer memory.
driver	Software unique to the device or type of device, and includes the set of commands the device accepts.
duty cycle	The ratio of the duration (time) that a signal is on to the total period of the signal.

E

EEPROM	Electrically erased programmable read-only memory. Read-only memory that you can erase with an electrical signal and reprogram.

F

fall time	The time for a signal to move from 90% to 10% of the signal value.
Fast Fourier Transform	An efficient mathematical algorithm used for spectrum analysis.
FFT	*See* Fast Fourier Transform.
FieldPoint	A family of industrial I/O modules from National Instruments.
filtering	Type of signal conditioning that allows you to filter unwanted signals from the signal you are trying to measure.
floating signal sources	Signal sources with voltage signals that are not connected to an absolute reference or system ground. Some common examples of floating signal sources are batteries, transformers, or thermocouples. *Also called* nonreferenced signal sources.
Fourier Transform	A mathematical technique that resolves a given signal into the sum of sines and cosines. Widely used as the FFT (Fast Fourier Transform), which is the basis for spectrum analysis.
frequency	f—the basic unit of rate, measured in events or oscillations per second using a frequency counter or spectrum analyzer. Frequency is the reciprocal of the period of a signal.
frequency response	The gain and phase response of a circuit or other unit under test at all frequencies of interest. Although the formal definition of frequency response includes both the gain and phase, in common usage, the frequency response often only implies the magnitude (gain). The frequency response is defined as the inverse Fourier Transform of the Impulse Response of a system.

G

gain	Amplification or attenuation of a signal.
GATE input pin	Counter input pin that controls when counting in your application occurs.
General Purpose Interface Bus	GPIB—synonymous with HP-IB. The standard bus used for controlling electronic instruments with a computer. Also called IEEE 488 bus because it is defined by ANSI/IEEE Standards 488-1978, 488.1-1987, and 488.2-1992.

GPIB	*See* General Purpose Interface Bus.
grounded signal sources	Signal sources with voltage signals that are referenced to a system ground, such as a building ground. *Also called* referenced signal sources.

H

handshaking	A type of protocol that makes it possible for two devices to synchronize operations.
Hz	Hertz. Cycles per second.

I

input range	Difference between the maximum and minimum voltages an analog input channel can measure at a gain of 1. The input range is a scalar value, not a pair of numbers. By itself, the input range does not uniquely determine the upper and lower voltage limits. An input range of 10 V could mean an upper limit of +10 V and a lower limit of 0 V or an upper limit of +5 V and a lower limit of –5 V. The combination of input range, polarity, and gain determines the input limits of an analog input channel. For some products, jumpers set the input range and polarity, although you can program them for other products. Most products have programmable gains. When you use SCXI modules, you also need their gains to determine the input limits.
instrument driver	A set of high-level functions that control and communicate with instrument hardware in a system.
interrupt	Signal that indicates that the central processing unit should suspend its current task to service a designated activity.
I/O	Input/Output. The transfer of data to or from a computer system involving communications channels, operator input devices, and/or data acquisition and control interfaces.
isolation	Type of signal conditioning in which you isolate the transducer signals from the computer for safety purposes. This protects you and the computer from large voltage spikes and makes sure the measurements from the DAQ device are not affected by differences in ground potentials.

IVI	Interchangeable Virtual Instruments—a software standard for creating a common interface (API) to common test and measurement instruments.
IVI driver	A driver written according to the IVI specification. The generic driver for a class of instruments (such as voltmeters) is called a class driver, whereas the driver for a specific instrument from a specific manufacturer is called a device-specific driver.

K

kH	Kilohertz.

L

Legacy MIO device	Devices, such as the AT-MIO-16, that typically are configured with jumpers and switches and are not Plug and Play compatible. They also use the 9513 type counter/timer chip.
limit settings	Maximum and minimum voltages of the analog signals you are measuring or generating.
linearization	Type of signal conditioning in which LabVIEW linearizes the voltage levels from transducers so the voltages can be scaled to measure physical phenomena.
LSB	Least Significant Bit.
lowpass filter	A circuit that attenuates the high-frequency components in an analog signal and only passes low frequencies. For imaging, a lowpass filter removes detail and blurs the image.

M

mA	Milliamp.
MAX	Measurement & Automation Explorer—A controlled, centralized configuration environment that allows you to configure NI devices.
MB	Megabytes of memory. 1 MB is equal to 1,024 KB.

measurement device	DAQ devices such as the E Series multifunction I/O (MIO) devices, SCXI signal conditioning modules, and switch modules.
MHz	Megahertz.
module	A board assembly and its associated mechanical parts, front panel, optional shields, and so on. A module contains everything required to occupy one or more slots in a mainframe. SCXI and PXI devices are modules.
multithreading	A technique for an operating system to handle multiple small tasks at one time, known as threads.

N

NI-DAQ	Driver software included with all NI measurement devices. NI-DAQ is an extensive library of VIs and functions you can call from an application development environment (ADE), such as LabVIEW, to program all the features of an NI measurement device, such as configuring, acquiring and generating data from, and sending data to the device.
NI-DAQ 7.0	Includes two NI-DAQ drivers—Traditional NI-DAQ and NI-DAQmx—each with its own API, hardware configuration, and software configuration.
NI-DAQmx	The latest NI-DAQ driver with new VIs, functions, and development tools for controlling measurement devices. The advantages of NI-DAQmx over earlier versions of NI-DAQ include the DAQ Assistant for configuring channels and measurement tasks for your device for use in LabVIEW, LabWindows/CVI, and Measurement Studio; increased performance such as faster single-point analog I/O; and a simpler API for creating DAQ applications using fewer functions and VIs than earlier versions of NI-DAQ.
NIST	National Institute of Standards and Technology—a federal technology agency that develops and promotes measurement, standards, and technology.
Non-referenced single-ended (NRSE) measurement system	All measurements are made with respect to a common reference, but the voltage at this reference can vary with respect to the measurement system ground.
ns	Nanoseconds.

Nyquist frequency	When an analog signal is sampled at a rate more than twice that of its highest frequency component, it can be properly reconstructed when reconverted back to the analog domain. The required sampling rate is called the Nyquist frequency.
Nyquist Theorem	A law of sampling theory stating that if a continuous bandwidth-limited signal contains no frequency components higher than half the frequency at which it is sampled, then the original signal can be recovered without distortion. If the signal is not sampled fast enough, aliasing will occur.

O

OUT output pin	Counter output pin where the counter can generate various TTL pulse waveforms.

P

PCI	Peripheral Component Interconnect. An industry-standard, high-speed databus.
period	The period of a signal, most often measured from one zero crossing to the next zero crossing of the same slope. The period of a signal is the reciprocal of its frequency (in Hz). Period is designated by the symbol T.
physical channel	*See* channel.
Poisson's Ratio	The negative ratio of the transverse strain to the longitudinal strain.
Property Node	Sets or finds the properties of a VI or application.
pulse	A signal whose amplitude deviates from zero for a short period of time.
pulse train	Multiple pulses.
pulse width	The time from the rising to the falling slope of a pulse (at 50% amplitude).
PXI	PCI eXtensions for Instrumentation. A modular, computer-based instrumentation platform.

Q

quantization error	The inherent uncertainty in digitizing an analog value due to the finite resolution of the conversion process. The quantization error depends on the number of bits in the converter, along with its errors, noise, and nonlinearities.

R

R	Resistance.
referenced single-ended (RSE) measurement system	All measurements are made with respect to a common reference or a ground. *Also called* a grounded measurement system.
rise time	The time for the signal to transition from 10% to 90% of the maximum signal of amplitude.
RMS	Root Mean Square.
RTD	Resistance temperature detector—A metallic probe that measures temperature based upon its coefficient of resistivity.
RTSI	Real-Time System Integration bus. The National Instruments timing bus that interconnects data acquisition devices directly by means of connectors on top of the devices for precise synchronization of functions.

S

S	Sample.
sampling period	Time interval between observations in a periodic sampling control system.
scan	One or more analog or digital input samples. Typically, the number of input samples in a scan equals the number of channels in the input group. For example, one pulse from the scan clock produces one scan that acquires one new sample from every analog input channel in the group.
scan clock	Clock that controls the time interval between scans in Traditional NI-DAQ. On products with interval scanning support (for example, E Series devices), this clock gates the channel clock on and off. On products with simultaneous sampling (for example, S Series devices), this clock determines the rate at which conversions are made across all channels.

scan rate	Number of times, or scans, per second that LabVIEW acquires data from channels. For example, at a scan rate of 10 Hz, LabVIEW samples each channel in a group 10 times per second.
SCPI	Standard Commands for Programmable Instruments—an extension of the IEEE 488.2 standard that defines a standard programming command set and syntax for device-specific operations.
SCXI	Signal Conditioning eXtensions for Instrumentation. The National Instruments product line for conditional low-level signals within an external chassis near sensors, so only high-level signals in a noisy environment are sent to data acquisition boards.
sec	Seconds.
sensor	Device that produces a voltage or current output representative of a physical property being measured, such as speed, temperature, or flow.
settling time	Amount of time required for a voltage to reach its final value within specified limits.
signal conditioning	Manipulation of signals to prepare them for digitizing.
signal-to-noise ratio	The ratio of total signal to noise expressed in decibels (dB). The larger the number, the better. SNR is calculated by SNR = 20 log (SignalRMS/NoiseRMS). This can be a peak, RMS, or other amplitude that appropriately characterizes the data.
SOURCE input pin	Counter input pin where the counter counts the signal transitions.
SNR	*See* signal-to-noise ratio.
strain gage	Thin conductor, which is attached to a material, that detects stress or vibrations in that material.

T

task	A collection of one or more channels, timing, triggering, and other properties in NI-DAQmx. A task represents a measurement or generation you want to perform.
ticks	Time in milliseconds required for the entire calculation.

Traditional NI-DAQ	An upgrade to the earlier version of NI-DAQ. Traditional NI-DAQ has the same VIs and functions and works the same way as NI-DAQ 6.9.*x*. You can use both Traditional NI-DAQ and NI-DAQmx on the same computer, which is not possible with NI-DAQ 6.9.*x*.
transducer excitation	Type of signal conditioning that uses external voltages and currents to excite the circuitry of a signal conditioning system into measuring physical phenomena.
trigger	Any event that causes or starts some form of data capture.
TTL	Transistor-Transistor Logic. A digital circuit composed of bipolar transistors wired in a certain manner.

U

update	One or more analog or digital output samples. Typically, the number of output samples in an update equals to the number of channels in the output group. For example, one pulse from the update clock produces one update that sends one new sample to every analog output channel in the group.
update rate	Number of output updates per second.

V

V	Volts.
VAC	Volts, Alternating Current.
VDC	Volts, Direct Current.
virtual channel	*See* channel.
Virtual Instrument Software Architecture	Single interface library for controlling GPIB, VXI, RS-232, and other types of instruments.
virtual instrumentation	A combination of hardware and/or software elements, typically used with a PC, that has the functionality of a classic stand-alone instrument.
VISA	*See* Virtual Instrument Software Architecture.
VXI	VME eXtensions for Instrumentation.

W

waveform	Multiple voltage readings taken at a specific sampling rate.
Wheatstone bridge	A technique for measuring voltage or resistance changes. The voltage between the midpoints of two parallel voltage dividers is measured, and the value of one of the resistors in the bridge is adjusted to give a balance voltage of zero. Changes in the bridge voltage, resulting from a resistive transducer as one of the elements in the bridge, are then measured. If a parallel resistor is the bridge in part of the transducers, temperature sensitivities are cancelled out.
window	A technique for selecting and scaling a block of data to be measured so that the data starts and stops in a way that optimizes the desired measurement result. Use a tapered start and stop of data to minimize edge effects that result in spectral leakage (a reduction in spectral resolution.) Common windows are Blackman, Blackman-Haris, Cosine, Exact Blackman, exponential, Flat Top, Force Hamming, Hanning, Kaiser-Bessel, and Triangle. The most frequently used window is the Hanning window.

Index