

Data Acquisition and Signal Conditioning Course Manual

Course Software Version 7.0
August 2003 Edition
Part Number 320733K-01

Copyright

© 1995–2003 National Instruments Corporation. All rights reserved.

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

Trademarks

CVI™, DAQ-STC™, FieldPoint™, IMAQ™, IVI™, LabVIEW™, Measurement Studio™, National Instruments™, NI™, NI-Motion™, ni.com™, NI-DAQ™, NI-IMAQ™, NI-PGIA™, RTSI™, and SCXI™ are trademarks of National Instruments Corporation. Product and company names mentioned herein are trademarks or trade names of their respective companies.

Patents

For patents covering National Instruments products, refer to the appropriate location: **Help»Patents** in your software, the `patents.txt` file on your CD, or ni.com/legal/patents.

Worldwide Technical Support and Product Information

ni.com

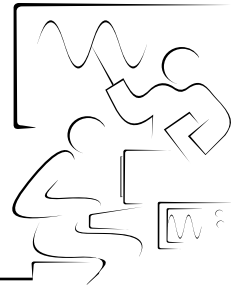
National Instruments Corporate Headquarters

11500 North Mopac Expressway Austin, Texas 78759-3504 USA Tel: 512 683 0100

Worldwide Offices

Australia 1800 300 800, Austria 43 0 662 45 79 90 0, Belgium 32 0 2 757 00 20, Brazil 55 11 3262 3599,
Canada (Calgary) 403 274 9391, Canada (Montreal) 514 288 5722, Canada (Ottawa) 613 233 5949,
Canada (Québec) 514 694 8521, Canada (Toronto) 905 785 0085, Canada (Vancouver) 514 685 7530, China 86 21 6555 7838,
Czech Republic 420 2 2423 5774, Denmark 45 45 76 26 00, Finland 385 0 9 725 725 11, France 33 0 1 48 14 24 24,
Germany 49 0 89 741 31 30, Greece 30 2 10 42 96 427, India 91 80 51190000, Israel 972 0 3 6393737, Italy 39 02 413091,
Japan 81 3 5472 2970, Korea 82 02 3451 3400, Malaysia 603 9131 0918, Mexico 001 800 010 0793,
Netherlands 31 0 348 433 466, New Zealand 0800 553 322, Norway 47 0 66 90 76 60, Poland 48 0 22 3390 150,
Portugal 351 210 311 210, Russia 7 095 783 68 51, Singapore 65 6226 5886, Slovenia 386 3 425 4200,
South Africa 27 0 11 805 8197, Spain 34 91 640 0085, Sweden 46 0 8 587 895 00, Switzerland 41 56 200 51 51,
Taiwan 886 2 2528 7227, Thailand 662 992 7519, United Kingdom 44 0 1635 523545

Contents



Student Guide

A. About This Manual	ix
B. What You Need to Get Started	x
C. Installing the Course Software.....	xi
D. Course Goals and Non-Goals	xii
E. Course Conventions	xiii

Lesson 1

Overview of Transducers, Signals, and Signal Conditioning

A. DAQ System Overview	1-2
B. Transducers	1-3
C. Signals.....	1-4
D. Signal Conditioning Overview	1-12
Summary	1-17

Lesson 2

Data Acquisition Hardware and Software

A. DAQ Hardware Overview	2-2
B. Components of a DAQ Device	2-7
C. Configuration Considerations	2-13
D. Grounding Issues	2-19
E. Types of Measurement Systems	2-21
F. Measuring Signal Sources	2-26
G. DAQ Software	2-33
H. NI-DAQ	2-34
I. Measurement & Automation Explorer	2-36
J. Overview of NI-DAQmx VIs	2-49
K. NI-DAQmx Task State Model	2-53
Summary	2-58

Lesson 3

Triggering

A. Triggering	3-2
Summary	3-11

Lesson 4**Analog Input**

A. Analog Input	4-2
B. Anti-aliasing Filters	4-7
C. Using the DAQmx Read VI.....	4-11
D. DAQ Device Architectures	4-17
E. Multiple-Point (Buffered) Analog Input.....	4-23
F. Continuous Acquisition Flowchart	4-33
Summary	4-49

Lesson 5**Signal Conditioning**

A. Overview of Signal Conditioning	5-2
B. Signal Conditioning Configuration.....	5-3
C. Signal Conditioning Functions	5-11
D. Filtering.....	5-21
E. Isolation	5-33
F. Transducer Conditioning	5-36
G. Thermocouples.....	5-37
H. Strain.....	5-46
I. Strain Gauge	5-47
J. Signal Conditioning for Strain Gauges.....	5-53
K. Strain Gauge Equations	5-56
Summary	5-64

Lesson 6**Signal Processing**

A. Discrete Fourier Transform (DFT) and Fast Fourier Transform (FFT).....	6-2
B. Frequency Spacing and Symmetry of the DFT/FFT	6-6
C. Power Spectrum.....	6-11
D. About Spectral Leakage and Smoothing Windows	6-19
E. Characteristics of Different Types of Window Functions	6-23
F. Determining Which Type of Window To Use.....	6-30
G. Filtering.....	6-36
H. Ideal Filters	6-37
I. Practical (Nonideal) Filters	6-39
J. Advantages of Digital Filters over Analog Filters.....	6-41
K. IIR and FIR Filters.....	6-42
L. Infinite Impulse Response Filters	6-44
M. IIR Filter Comparison.....	6-50
N. Transient Response of IIR Filters	6-52
O. Finite Impulse Response Filters.....	6-60
Summary	6-62

Lesson 7**Analog Output**

A. Analog Output Architecture.....	7-2
B. Using the DAQmx Write VI.....	7-4
C. Multiple-Point (Buffered) AO VIs	7-10
D. Finite Buffered Generation	7-11
E. Continuous Buffered Generation	7-21
Summary	7-32

Lesson 8**Digital I/O**

A. Digital Signals.....	8-2
B. Digital I/O	8-4
Summary	8-12

Lesson 9**Counters**

A. Counter Signals.....	9-2
B. Counter Chips	9-5
C. Counter I/O	9-6
D. Edge Counting	9-8
E. Advanced Edge Counting	9-12
F. Pulse Generation	9-17
G. Pulse Measurement	9-25
H. Frequency Measurements	9-30
I. Position Measurement.....	9-35
Summary	9-40

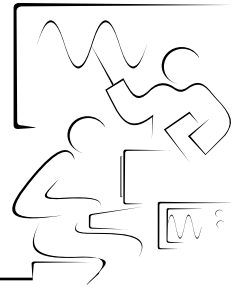
Lesson 10**Synchronization**

A. Explicit State Transitions.....	10-2
B. Single Device Synchronization.....	10-8
C. Multiple Device Synchronization	10-29
Summary	10-38

Appendix A

A. Theory of Common Transducers	A-2
B. Analog I/O Circuitry	A-8

Student Guide

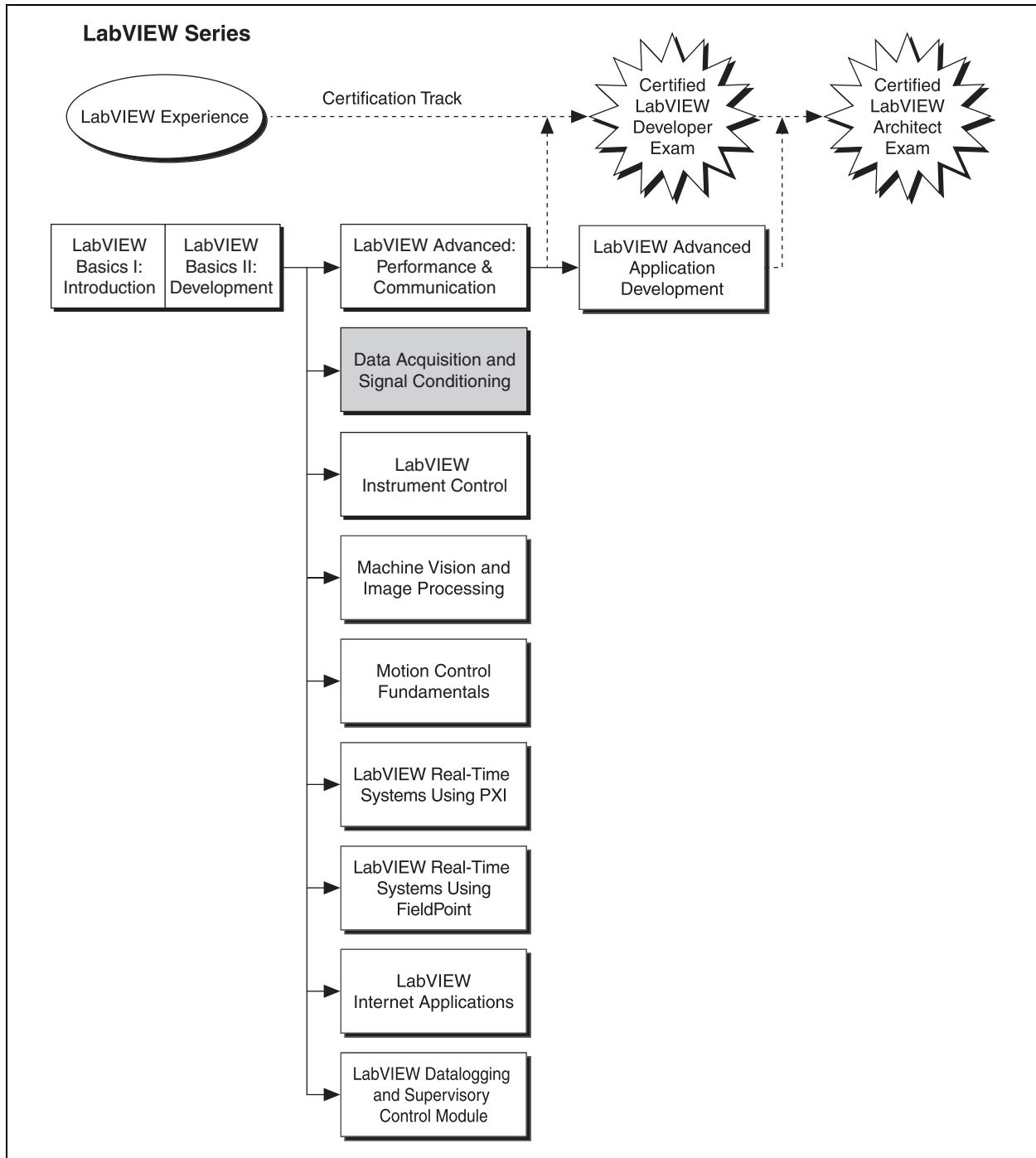


Thank you for purchasing the *Data Acquisition and Signal Conditioning* course kit. This course manual and accompanying software are used in the three-day, hands-on *Data Acquisition and Signal Conditioning* course.



Note You can apply the full purchase of this course kit toward the corresponding course registration fee if you register within 90 days of purchasing the kit. Visit ni.com/training for online course schedules, syllabi, training centers, and class registration.

The *Data Acquisition and Signal Conditioning* course is part of a series of courses designed to build your proficiency with LabVIEW and help you prepare for exams to become an NI Certified LabVIEW Developer and NI Certified LabVIEW Architect. The following illustration shows the courses that are part of the LabVIEW training series. Refer to ni.com/training for more information about NI Certification.



A. About This Manual

This course manual teaches you the fundamentals of data acquisition (DAQ) and signal conditioning and introduces techniques you can implement in real-world applications.

This course also expands on good LabVIEW programming style issues from the *LabVIEW Basics I: Introduction* and *LabVIEW Basics II: Development* courses and discusses several programming guidelines you can use in your data acquisition and signal conditioning applications. This course manual assumes that you are familiar with Windows, that you have experience writing algorithms in the form of flowcharts or block diagrams, and that you have taken the *LabVIEW Basics I: Introduction* course or you have familiarity with all the concepts contained therein. This course also assumes you have approximately one year or more of LabVIEW development experience.

The course manual is divided into lessons, each covering a topic or set of topics. Each lesson consists of the following:

- An introduction that describes the purpose of the lesson and what you will learn
- A description of the topics in the lesson
- A set of exercises to reinforce those topics
- A summary that outlines important concepts and skills taught in the lesson

Several exercises in this manual use a plug-in multifunction DAQ device connected to a DAQ Signal Accessory, and either SCXI or SCC signal conditioning equipment.

B. What You Need to Get Started

Before you use this course manual, make sure you have all the following items:

- Computer running Windows 98 or later
- LabVIEW Full Development System or Professional Development System 7.0 or later



Note This course assumes you are using the default installation of LabVIEW. If you have changed the palette views from the default settings, some palette paths described in the course might not match your settings. To reset palette views to LabVIEW defaults, select **Tools»Options** and select **Controls/Functions Palettes** from the top pull-down menu. Set **Palette View** to **Express** and set **Format** to **Standard**. Click the **OK** button to apply the changes and close the dialog box.

- MIO Series DAQ board
- DAQ Signal Accessory and cable
- SCXI chassis, modules, terminal blocks, and transducers as described in Lesson 5, *Signal Conditioning*
- Data Acquisition and Signal Conditioning* course CD, containing the following files:

Directory	Description
Exercises	Contains all the VIs and support files needed to complete the exercises in this course
Solutions	Contains completed versions of the VIs you build in the exercises for this course

C. Installing the Course Software

To install the software used for this course, insert the Data Acquisition and Signal Conditioning course disk and install the required files in the following directory structure:

- All exercise VIs and associated support files are installed in the C:\Exercises\LabVIEW DAQ directory.
- All exercise solutions are installed in the C:\Solutions\LabVIEW DAQ directory.

D. Course Goals and Non-Goals

The purpose of this course is to teach you the components of a DAQ system and to teach you how to use that system. By the end of this course, you will understand the five components of a DAQ system:

- Transducers
- Signals
- Signal conditioning
- DAQ hardware
- DAQ software

You also will know how to use LabVIEW with a DAQ device to:

- Acquire analog signals
- Perform signal conditioning on acquired signals
- Generate analog signals
- Perform non-timed digital input and output
- Use counters for event counting, pulse generation, pulse measurement, and frequency measurement

This course does *not* describe any of the following:

- Basic principles of LabVIEW covered in the *LabVIEW Basics I: Introduction* course
- Every built-in VI, function, or object; refer to the *LabVIEW Help* for more information about LabVIEW features not described in this course
- Developing a complete application for any student in the class; refer to the NI Example Finder, available by selecting **Help»Find Examples**, for example VIs you can use and incorporate into VIs you create

Refer to the *LabVIEW Help* for more information on a particular DAQ Library VI.

If you are seeking help for developing your application, please discuss this need with your instructor outside of class time or contact National Instruments Technical Support. For further information on technical support, refer to the National Instruments Web site at ni.com.

E. Course Conventions

The following conventions are used in this course manual:

» The » symbol leads you through nested menu items and dialog box options to a final action. The sequence **File»Page Setup»Options** directs you to pull down the **File** menu, select the **Page Setup** item, and select **Options** from the last dialog box.



This icon denotes a tip, which alerts you to advisory information.



This icon denotes a note, which alerts you to important information.



This icon denotes a caution, which advises you of precautions to take to avoid injury, data loss, or a system crash.

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, controls and buttons on the front panel, dialog boxes, sections of dialog boxes, menu names, and palette names.

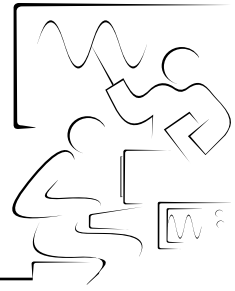
italic Italic text denotes variables, emphasis, a cross reference, or an introduction to a key concept. This font also denotes text that is a placeholder for a word or value that you must supply.

`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.

monospace italic Italic text in this font denotes text that is a placeholder for a word or value that you must supply.

Lesson 1

Overview of Transducers, Signals, and Signal Conditioning



This lesson introduces the basics of data acquisition (DAQ).

You Will Learn:

- A. The basics of data acquisition (DAQ)
- B. The basics of transducers
- C. The basics of signals
- D. The basics of signal conditioning

A. DAQ System Overview

The purpose of a DAQ system is to measure a physical phenomenon such as light, temperature, pressure, or sound. A DAQ system includes the following building blocks:

- Transducer
- Signal
- Signal conditioning
- DAQ device
- Driver level and application level software

With these five building blocks, you can bring the physical phenomenon you want to measure into the computer for analysis and presentation.

B. Transducers

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. The following table lists some common transducers.

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

Types of Transducers

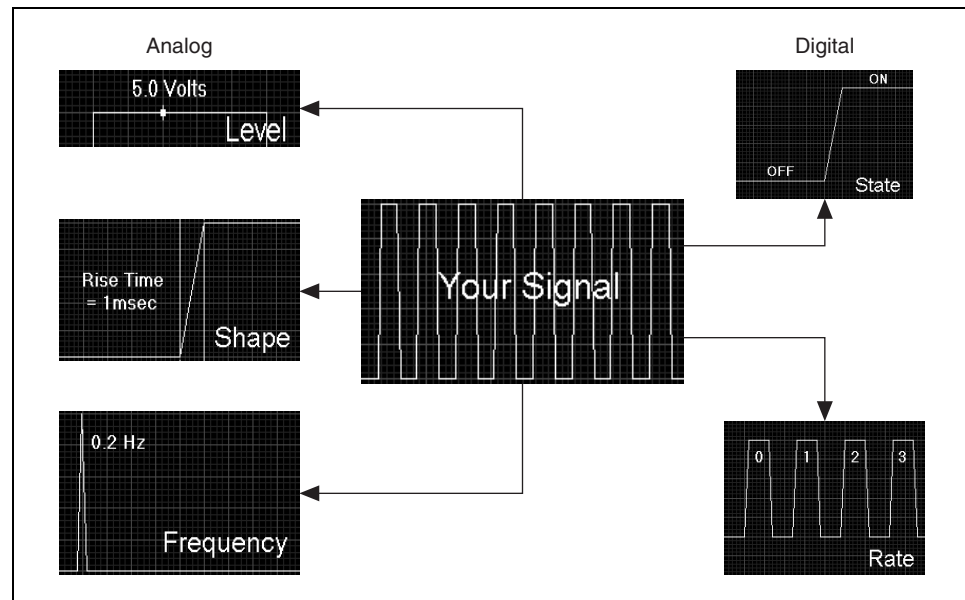
Transducers are used for a variety of needs such as measuring temperature, pressure, and fluid flow. Refer to ni.com/sensors for more information about transducers and where to obtain them.

Different transducers have different requirements for converting a physical phenomenon into a measurable signal. For example, a resistance temperature detector (RTD) needs an excitation current to measure the temperature. A thermocouple does not need excitation current, but it does need cold-junction compensation. Strain gauges use a configuration of resistors called a Wheatstone bridge to measure strain. Before you set up the system, you should know if the transducer has any special requirements. Contact the vendor of the transducer for more information about how to properly use the transducer.

C. Signals

With the help of a transducer, you convert a physical phenomenon into a signal. Not all signals are measured in the same manner, so you need to categorize the signal as digital or analog.

After you categorize the signal, decide which type of information you want from that signal. The possible types of information you can obtain from a signal are state, rate, level, shape, and frequency.



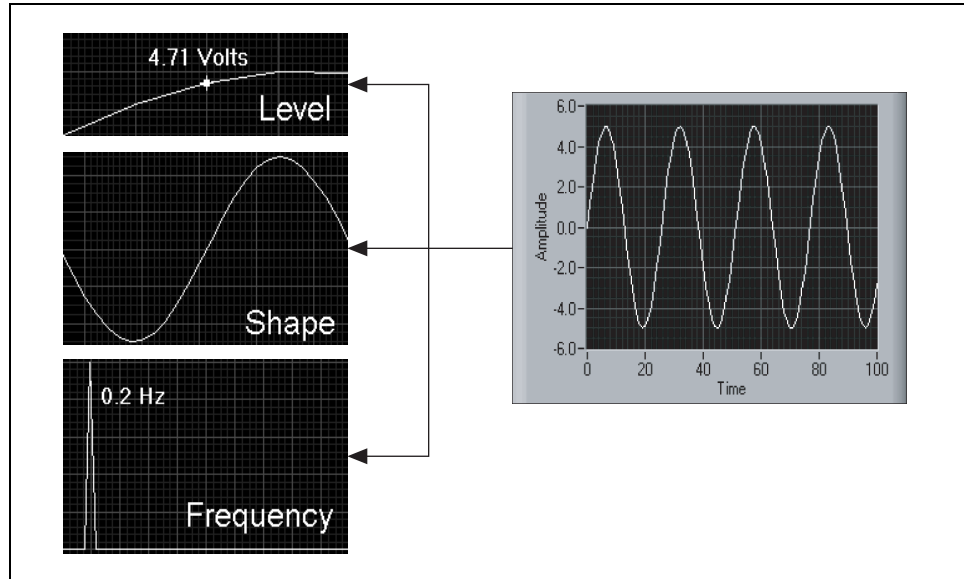
Note This discussion of signals assumes that you are acquiring the signal. However, most of the points also apply to generating a signal. The difference between acquiring and generating a signal is that you do not need analysis when you generate a signal with a specific frequency.

Analog Signals

Unlike digital signals, an analog signal can be at any voltage level with respect to time. Because an analog signal can be at any state at any time, the physical aspects you want to measure differ from those of a digital signal.

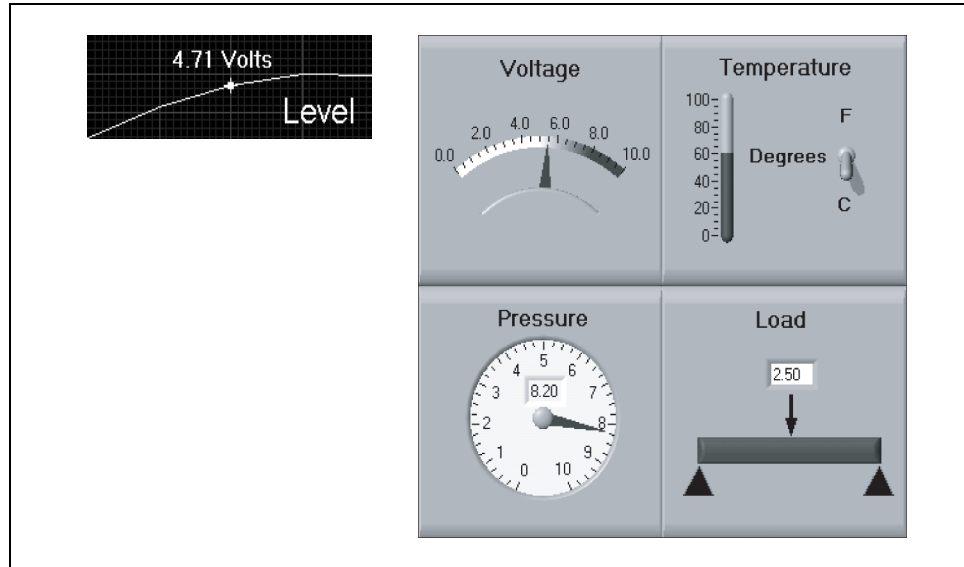
Analog Signal Information

You can measure the level, shape, and frequency of an analog signal, as shown in the following illustration.



- **Level**—Measuring the level of an analog signal is similar to measuring the state of a digital signal. The difference is that an analog signal can be at any voltage level, but a digital signal is either 0 V or 5 V.
- **Shape**—Measuring the shape of the signal is often important because analog signals can be at any state with respect to time. For example, a sine wave has a different shape than a sawtooth wave. Measuring the shape of a signal can lead to analysis of other aspects of the signal, such as peak values, slope, or integration.
- **Frequency**—Measuring the frequency of an analog signal is similar to measuring the rate of a digital signal. However, you cannot directly measure the frequency of an analog signal. You need to perform software analysis on the signal to extract the frequency information, usually with a Fourier Transform.

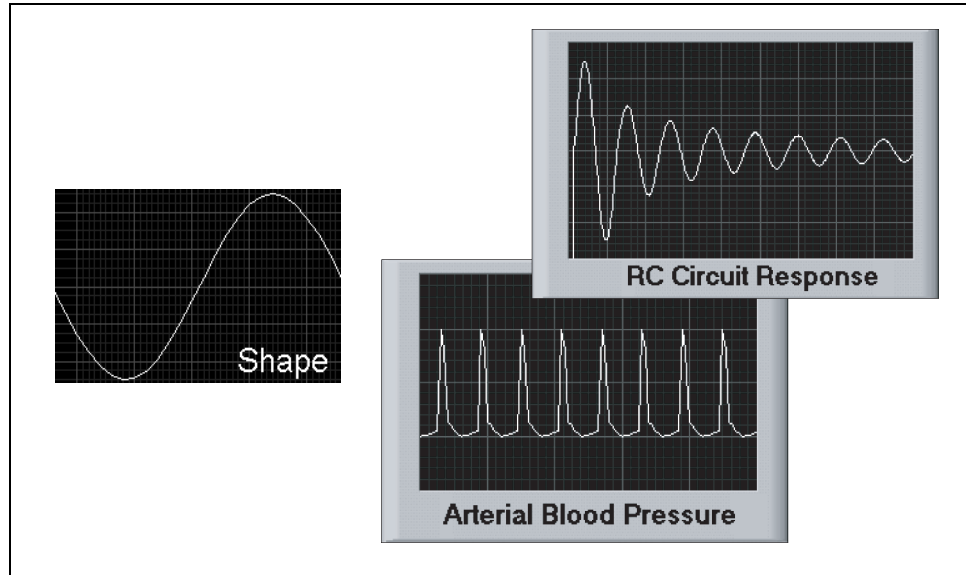
The level of most signals does not change much with respect to time. However, you usually need to measure the signal with a high level of accuracy. Therefore, you need a DAQ device with a high resolution, but not a high sample rate. Using a variety of transducers, you could measure the voltage of a power supply, the temperature of a mixing tank, the pressure inside a hose, or the load on a piece of machinery, as shown in the following figure.



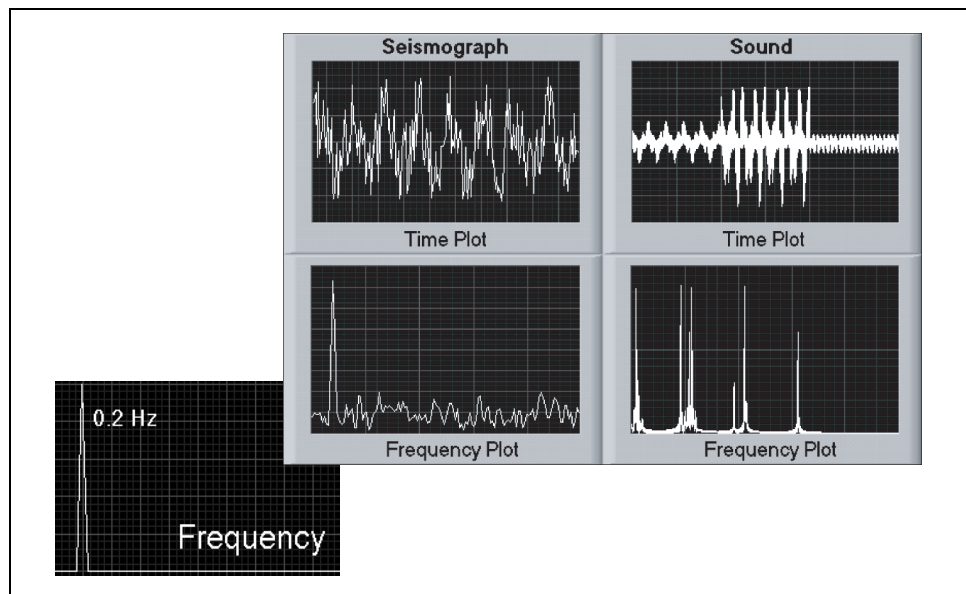
When you measure the shape of a signal, you need to know the relationship of that signal with respect to time. Some signals change rapidly with respect to time. Most applications in which you measure shape also need a high level of accuracy. Therefore, you need a DAQ device with a high resolution and a high sample rate.

Examples of measuring shape are abundant in the medical, electronic, and automotive industries, and they range from measuring a heartbeat, to measuring a video signal, to measuring the vibration of a spring. After you have acquired the signal, you can analyze it to extract the specific information you need about the shape.

For example, when you measure blood pressure, you are concerned with the peak value. However, with resistor-capacitor (RC) circuit response, you are more concerned with how the amplitude varies over time, as shown in the following figure.



When you measure the frequency of a signal, you need to know the relationship of that signal with respect to time. Many signals change rapidly with respect to time. Most applications in which you measure frequency also need a high level of accuracy. Therefore, you need a DAQ device with a high resolution and a high sample rate. Once you acquire the signal with respect to time (time plot), use software analysis to convert the time plot signal into the frequency plot. LabVIEW provides the necessary software analysis, as shown in the following figure.



Digital Signals

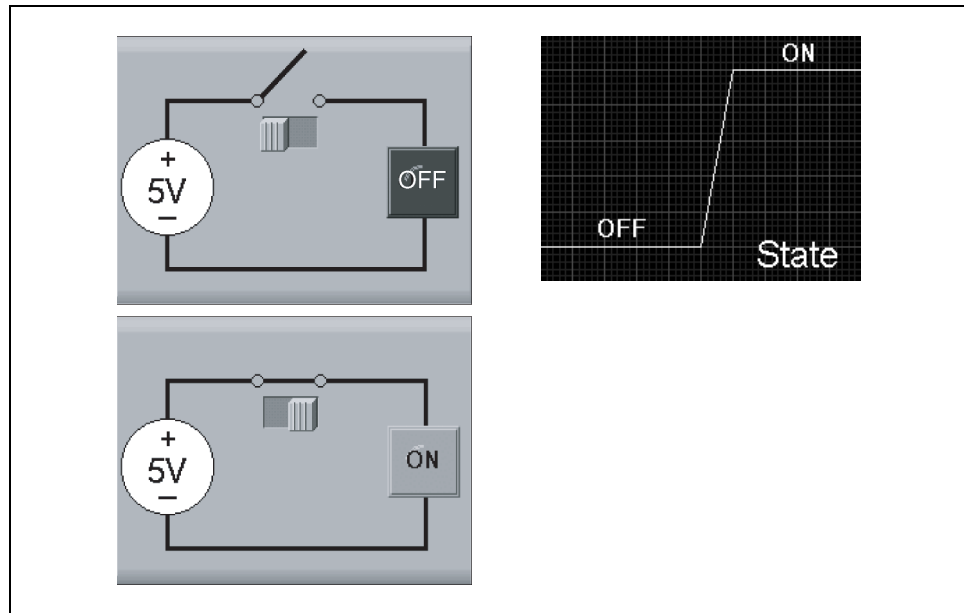
A digital signal has only two possible states—ON (logic high) or OFF (logic low). A digital signal is often referred to as a Transistor-to-Transistor Logic (TTL) signal. The specifications for a TTL signal define a voltage level between 0 and 0.8 V as logic low and a voltage level between 2 and 5 V as logic high. Most digital devices accept a TTL-compatible signal.

Digital Signal Information

You only can measure two aspects of a digital signal: state and rate.

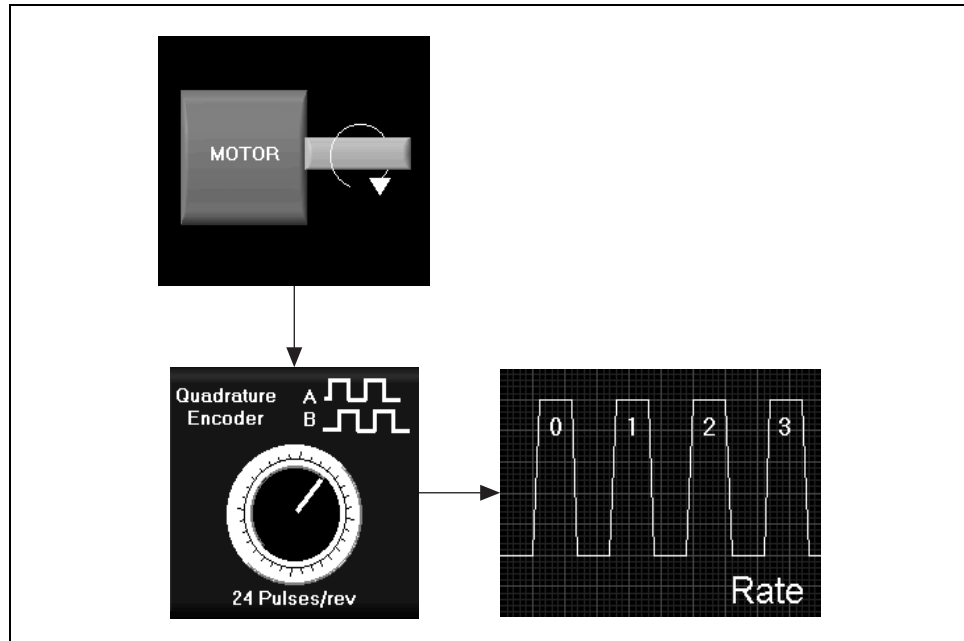
- **State**—A digital signal only has two possible states: ON or OFF. One of the aspects of a digital signal you can measure is if the state is ON or OFF.
- **Rate**—A digital signal also changes state with respect to time. The other aspect of a digital signal you can measure is the rate, or how the digital signal changes states over time.

Consider the following example of measuring the state of a digital signal.



Assume you have a switch that you want to monitor. This switch turns a light on or off. In the example above, when the switch is open, you measure 0 V (OFF). When the switch is closed, you measure 5 V (ON). By measuring the state of the digital signal, you can determine if the light is on or off.

Now look at the following example for measuring the rate of a digital signal.

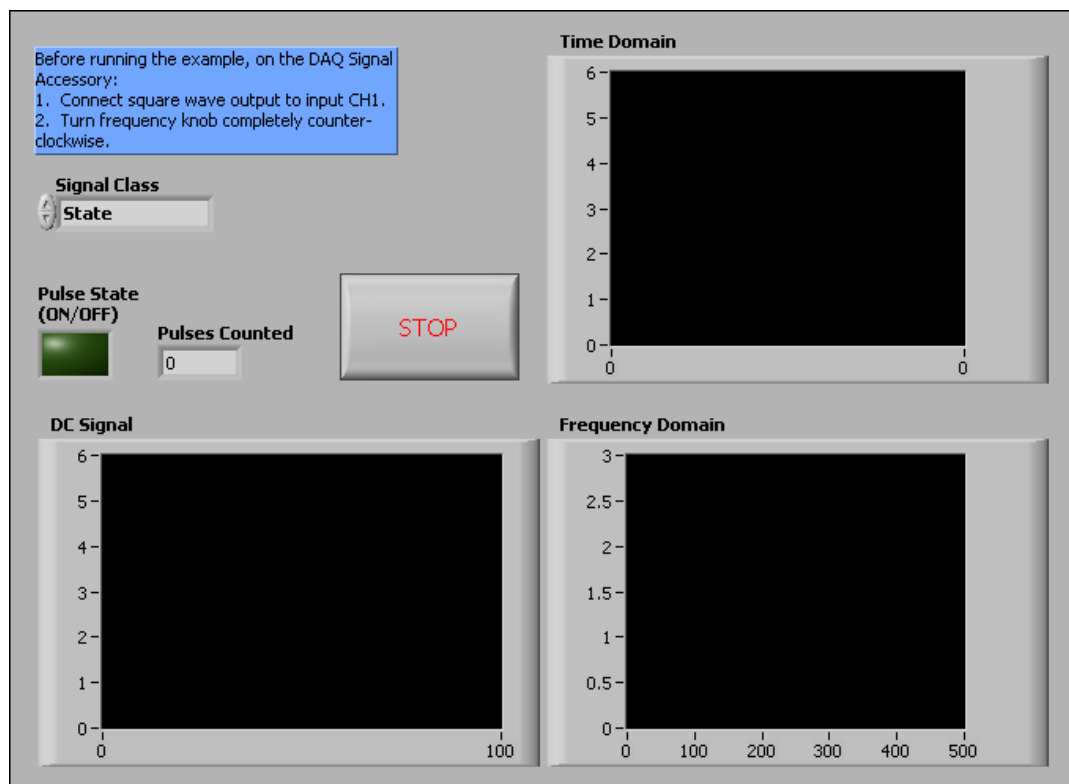


Assume you have a motor, and you want to determine how fast the shaft of the motor is spinning. An encoder is a transducer that can convert the rotary motion of the motor shaft into a digital signal. When an encoder rotates, it produces two digital signals. Each digital signal is a series of alternating on or off states, called a pulse train. For each increment of rotation, you get a pulse. The increment of rotation depends on the encoder. For example, the DAQ Signal Accessory you use in this course has an encoder that gives 24 pulses per revolution. You can measure the rate of one of the pulse trains to determine how fast the shaft is rotating. You can measure both pulse trains to determine not only how fast the shaft is rotating but also the direction in which it is rotating.

Exercise 1-1 Signals Example VI

Objective: To measure the state, rate, level, shape, and frequency of one signal.

1. Make the following connections and adjustments on the DAQ Signal Accessory:
 - a. Connect the square wave from the Function Generator to analog input 1.
 - b. Turn the **Frequency Adjust** knob as low as possible. Make sure that the **Frequency Range** is set on 100 Hz – 10 kHz.
2. Open the Signals Example VI located in the C:\Exercises\LabVIEW DAQ directory. This VI measures the state, rate, level, shape, and frequency of the square wave you connected to analog input 1, as shown in the following front panel.



3. Run the VI.
4. Change the **Signal Class** control to **Digital On/Off** by clicking the increment button to the left of the control. The Pulse State (ON/OFF) LED should flicker.



Note You are connecting the square wave to an analog input channel, not a digital line. Therefore, this example does not detect the state of the signal based on the TTL definition that 0 V to 0.8 V is OFF and 2 V to 5 V is ON. Instead, this example reads any value below 2.5 V as OFF and any value above 2.5 V as ON.

5. Change the **Signal Class** control to **Digital Pulse**. The **Pulses Counted** indicator should increase.



Note You have connected the square wave to an analog input channel, not a counter. Therefore, this example does not count the pulses by detecting the edge of a signal as a counter would. Instead, it increments the **Pulses Counted** indicator every time the signal rises above 2.5 V.

6. Change the **Signal Class** control to **Analog DC** by clicking the increment button to the left of the control. The DC signal chart should display points that are either at 0 V or 5 V. The chart does not display the shape or frequency of the signal, only the level. When you measure the level of a signal, the signal plots along a time axis, which you do not see when you measure the state of the signal. When you acquire the level of a signal, the speed of the acquisition is not as important as the accuracy of the measurement. Therefore, this example uses a software-timed acquisition to measure the level of the square wave.
7. Change the **Signal Class** control to **Time Domain** by clicking the increment button to the left of the control. The time-domain graph should display the pulse train. To acquire the shape of the signal, the acquisition rate must be very fast. To achieve an acquisition rate fast enough to acquire the shape of the square wave, this example uses a hardware-timed acquisition.
8. Change the **Signal Class** control to **Frequency Domain** by clicking the increment button to the left of the control. The frequency-domain graph should have a large spike at around 100 Hz. Remember that software analysis is required to obtain the frequency of the square wave. LabVIEW calculates the power spectrum of the square wave and plots it on the graph. A power spectrum is a way to measure the frequencies present in a signal.
9. Stop and close the VI. Do not save changes.

End of Exercise 1-1

D. Signal Conditioning Overview

You cannot always connect the signal directly to a DAQ device. You might need to alter the signal to make it suitable for a DAQ device to measure.

Signal conditioning is one of the most important technologies in your measurement and automation system. It provides the interface between your signals/sensors and measurement system. With National Instruments front-end signal conditioning, you integrate numerous technologies into one platform, increasing the number and types of measurements you can make. Depending on the acquisition speed, types of signals/sensors, and number of channels for your application, choose from National Instruments SCXI or SCC signal conditioning systems.

Choose Signal Conditioning eXtensions for Instrumentation (SCXI) platform and architecture when you need to automate a high-channel count application or develop a system requiring high acquisition rates. SCXI is a proven system in use by engineers and scientists since 1991 and includes more than 25 modules.

National Instruments Signal Conditioning Components (SCC) system is ideal for portable systems requiring low-to-medium channel counts. You can choose from more than 30 single- and dual-channel count modules.

Refer to ni.com/sigcon for more information about National Instruments SCXI products and other signal conditioning hardware.

Most transducers need some sort of external hardware to perform their job. For example, resistance temperature detectors need an excitation current, and strain gauges need a configuration of resistors called a Wheatstone bridge. 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 1-1 shows some common types of transducers and signals and the signal conditioning each requires.

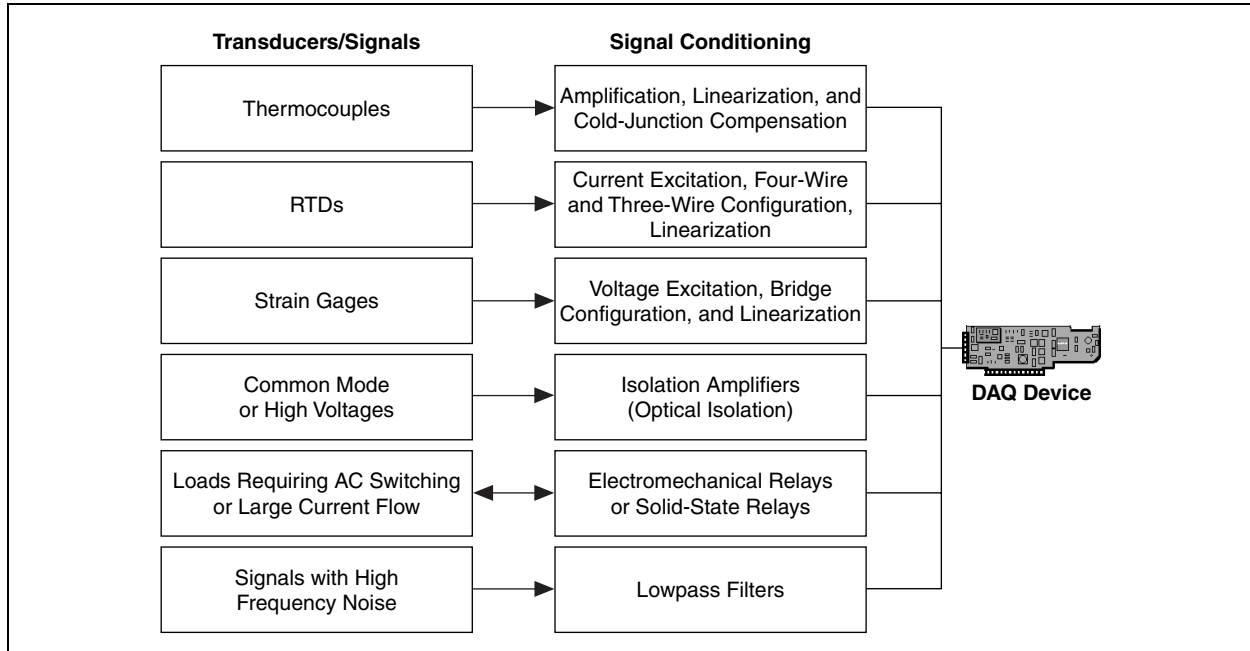


Figure 1-1. Common Types of Transducers/Signals and Signal Conditioning

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. A common example is the signal from a thermocouple, which outputs a voltage in the millivolt range.

If you send the signal from a thermocouple straight to a DAQ device, a change of a degree or two in temperature might not be detected by the system. However, if you amplify the signal, you have a signal that is better suited to the range of the DAQ device. You can amplify the signal either on the DAQ device or externally.

Amplify low-level signals at the DAQ device or SCXI module (labeled as the External Amplifier in Figure 1-2) 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).

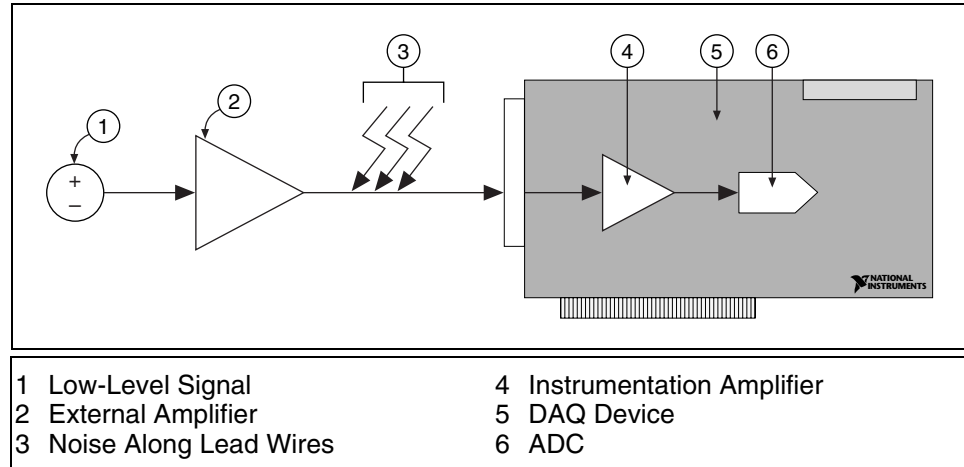


Figure 1-2. Amplifying Signals Near the Source to Increase Signal-to-Noise Ratio (SNR)

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 the 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.

Signal-to-Noise Ratio

The SNR is a measure of how much noise exists in a signal compared to the signal itself. SNR is defined as the voltage level of the signal divided by the voltage level of the noise. The larger the SNR, the better. As shown in the following table, the SNR is best when the signal is only amplified externally and worst when the signal is only amplified on the DAQ device.

	Signal Voltage	SCXI Amplification	Noise in Lead Wires	DAQ Device Amplification	Digitized Voltage	SNR
Amplify only at DAQ device	0.01 V	none	0.001 V	$\times 100$	1.1 V	10
Amplify at SCXI and DAQ device	0.01 V	$\times 10$	0.001 V	$\times 10$	1.01 V	100
Amplify only at SCXI	0.01 V	$\times 100$	0.001 V	none	1.001 V	1000

Other Types of Signal Conditioning

There are several other types of signal conditioning.

- **Linearization**—Many transducers have a nonlinear response to changes in the physical phenomena you measure. For example, a change in voltage of 10 mV for a thermocouple is usually not a change of 10 degrees. Most transducers have linearization tables that map out how to scale the transducer. The linearization of the transducer can be done either in hardware or software. 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 gauges, RTDs, thermocouples, and thermistors.
- **Transducer Excitation**—Signal conditioning systems can generate excitation, which some transducers require for operation. Strain gauges 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.

- **Filtering**—You can use filtering to remove unwanted portions of the signal. Most noise is created by AC power—for example, computer power supply or overhead lights. Noise appears at around 60 Hz. Use a lowpass filter with a cutoff frequency less than 60 Hz to remove noise from the signal. You can use filtering in hardware or software.

Summary

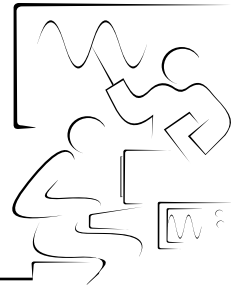
A DAQ system consists of five components:

- Transducers convert a physical phenomenon into a measurable signal.
- Signals can be either digital or analog. Depending on the signal, you can measure the state, rate, level, shape, or frequency.
- Signal conditioning makes signals easier to measure with a DAQ device.
- DAQ hardware
- DAQ software

Notes

Lesson 2

Data Acquisition Hardware and Software



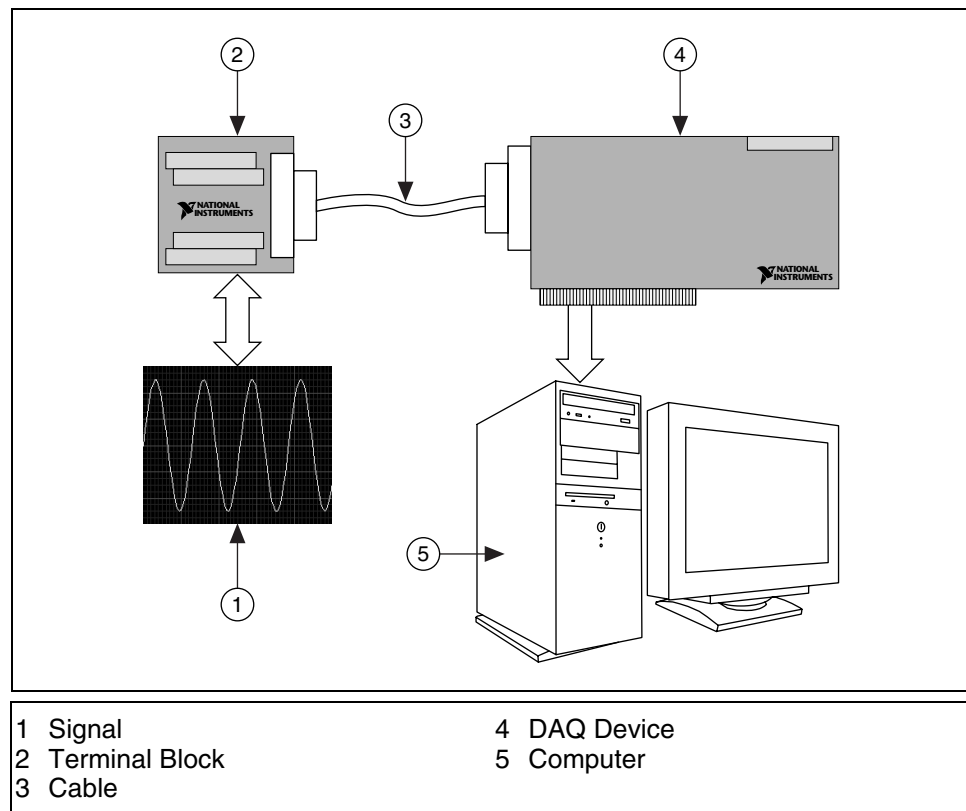
This lesson describes DAQ hardware and DAQ software.

You Will Learn:

- A. The basic hardware used in a DAQ system
- B. The components of a DAQ device and their uses
- C. Items to consider when you configure the device in software
- D. The software you can use to configure and control a DAQ system
- E. Overview of NI-DAQmx VIs and Property Nodes
- F. NI-DAQmx task state model

A. DAQ Hardware Overview

A typical DAQ system has three basic types of hardware—a terminal block, a cable, and a DAQ device, as shown in the following illustration. This section describes each type of hardware, then focuses on the components of a DAQ device and what each component does. You also will learn important considerations to keep in mind when you configure a DAQ device.

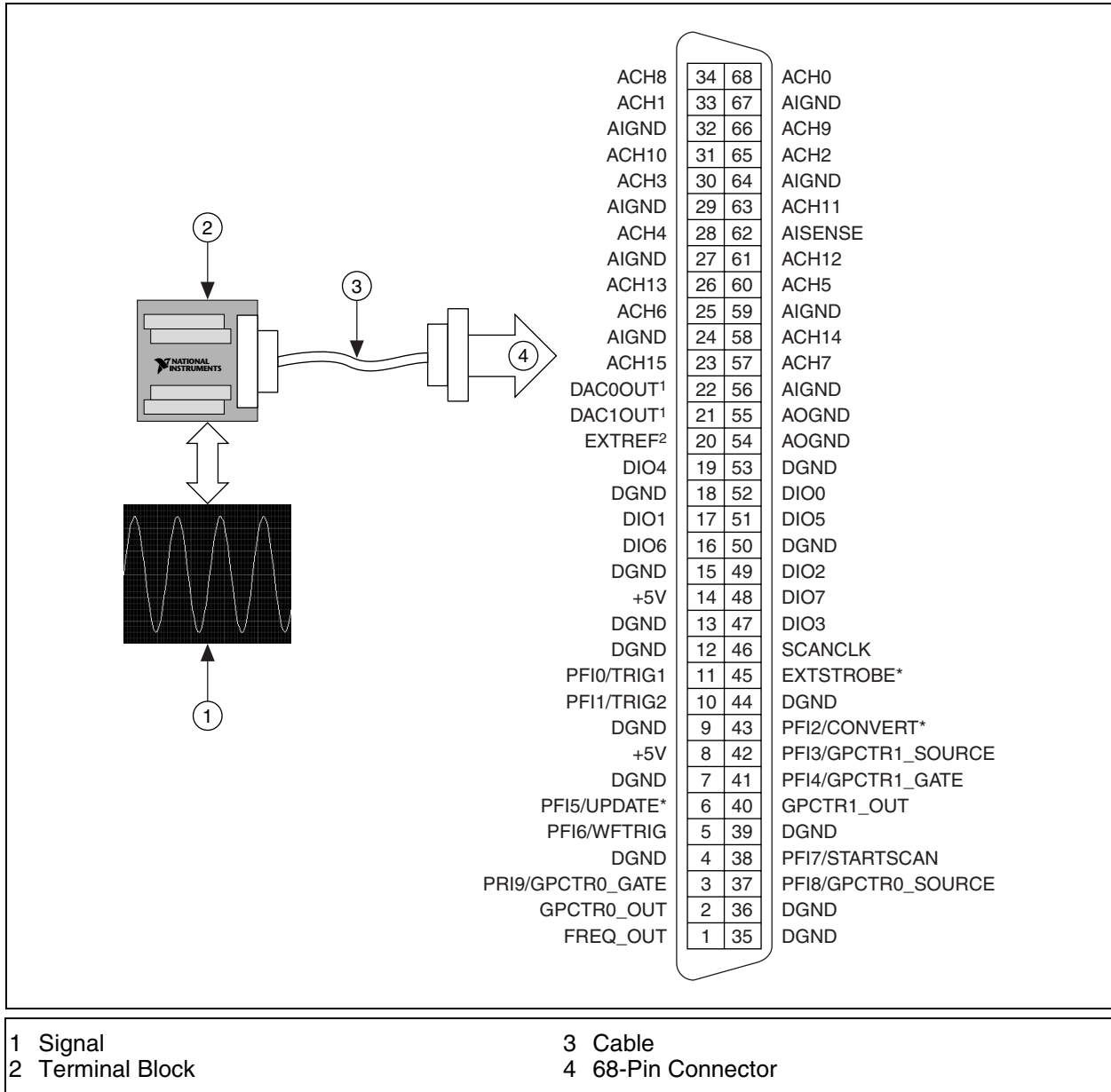


Now that you have converted a physical phenomenon into a measurable signal with or without signal conditioning, you need to acquire that signal. To acquire a signal, you need a terminal block, a cable, a DAQ device, and a computer. This hardware combination can transform a standard computer into a measurement and automation system.

Terminal Block and Cable

A terminal block provides a place to connect signals. It consists of screw or spring terminals for connecting signals and a connector for attaching a cable to connect the terminal block to a DAQ device. Terminal blocks have 100, 68, or 50 terminals. The type of terminal block you should choose depends on two factors—the device and the number of signals you are measuring. A terminal block with 68 terminals offers more ground terminals to connect a signal to than a terminal block with 50 terminals. Having more ground terminals prevents the need to overlap wires to reach a ground terminal,

which can cause interference between the signals. The following illustration shows the layout of the screw terminals. Terminal blocks can be shielded or non-shielded. Shielded terminal blocks offer better protection against noise. Some terminal blocks contain extra features, such as cold-junction compensation, that are necessary to measure a thermocouple properly.



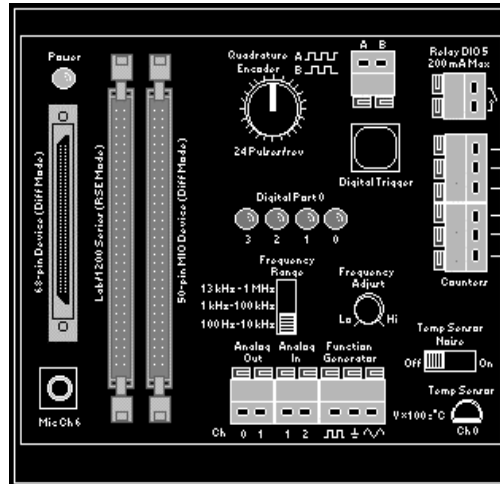
- 1 Signal
- 2 Terminal Block
- 3 Cable
- 4 68-Pin Connector

A cable transports the signal from the terminal block to the DAQ device. Cables come in 100-, 68-, and 50-pin configurations. Choose a configuration depending on the terminal block and the DAQ device you are using. Cables, like terminal blocks, are shielded or non-shielded.

Refer to the DAQ section of the National Instruments catalog or to ni.com/products for more information about specific types of terminal blocks and cables.

DAQ Signal Accessory

The following illustration shows the terminal block you are using for this course, the DAQ Signal Accessory.



The DAQ Signal Accessory is a customized terminal block designed for learning purposes. It has three different cable connectors to accommodate many different DAQ devices and spring terminals to connect signals. You can access three analog input channels, one of which is connected to the temperature sensor, and two analog output channels.

The DAQ Signal Accessory includes a function generator with a switch to select the frequency range of the signal, and a frequency knob. The function generator can produce a sine wave or a square wave. A connection to ground is located between the sine wave and square wave terminal.

A digital trigger button produces a TTL pulse for triggering analog input or output. When you press the trigger button, the signal goes from +5 V to 0 V when pressed and returns to +5 V when you release the button. Four LEDs connect to the first four digital lines on the DAQ device. The LEDs use reverse logic, so when the digital line is high, the LED is off and vice versa.

The DAQ Signal Accessory has a quadrature encoder that produces two pulse trains when you turn the encoder knob. Terminals are provided for the input and output signals of two counters on the DAQ device. The DAQ Signal Accessory also has a relay, a thermocouple input, and a microphone jack.

DAQ Device

Most DAQ devices have four standard elements: analog input, analog output, digital I/O, and counters. The most common National Instruments DAQ devices are the E Series devices.

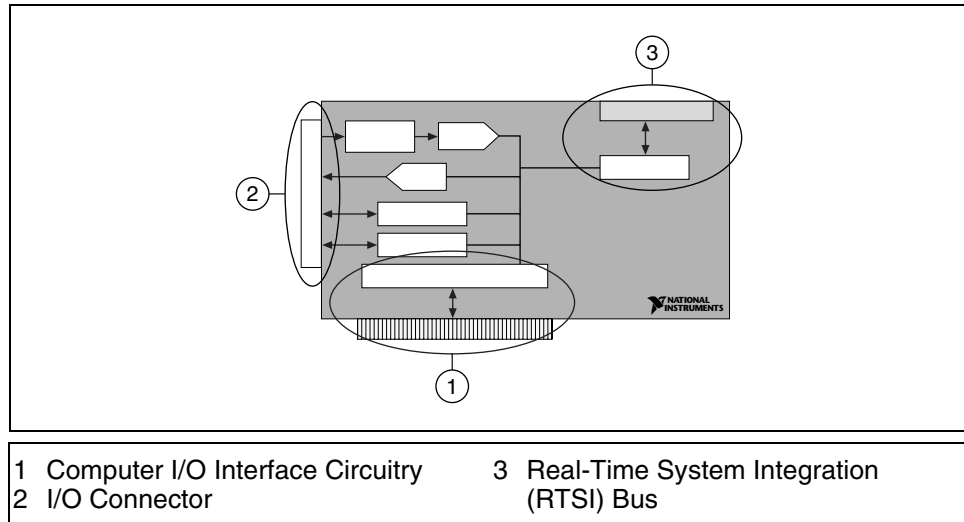
A typical E Series device consists of 16 analog input channels, two analog output channels, eight digital lines, and two counters. National Instruments also offers specialty devices for applications in which an E Series device is not applicable. For example, NI also makes high-speed digital devices that offer timed digital I/O, high-speed analog output devices for advanced waveform generation, and dynamic signal acquisition (DSA) devices for analyzing rapidly changing signals, such as vibration or sonar.

You can transfer the signal you measure with the DAQ device to the computer through a variety of different bus structures. For example, you can use a DAQ device that plugs into the PCI bus of a computer, a DAQ device connected to the PCMCIA socket of a laptop, or a DAQ device connected to the USB port of a computer.

You also can use PXI/CompactPCI to create a portable, versatile, and rugged measurement system. Refer to the DAQ section of the NI catalog or to ni.com/products for more information about specific types of DAQ devices.

B. Components of a DAQ Device

The following illustration shows the components of a DAQ device.



Interfaces

A typical DAQ device has three interfaces for receiving and sending signals: the I/O connector, the computer I/O interface circuitry, and the Real-Time System Integration (RTSI) Bus.

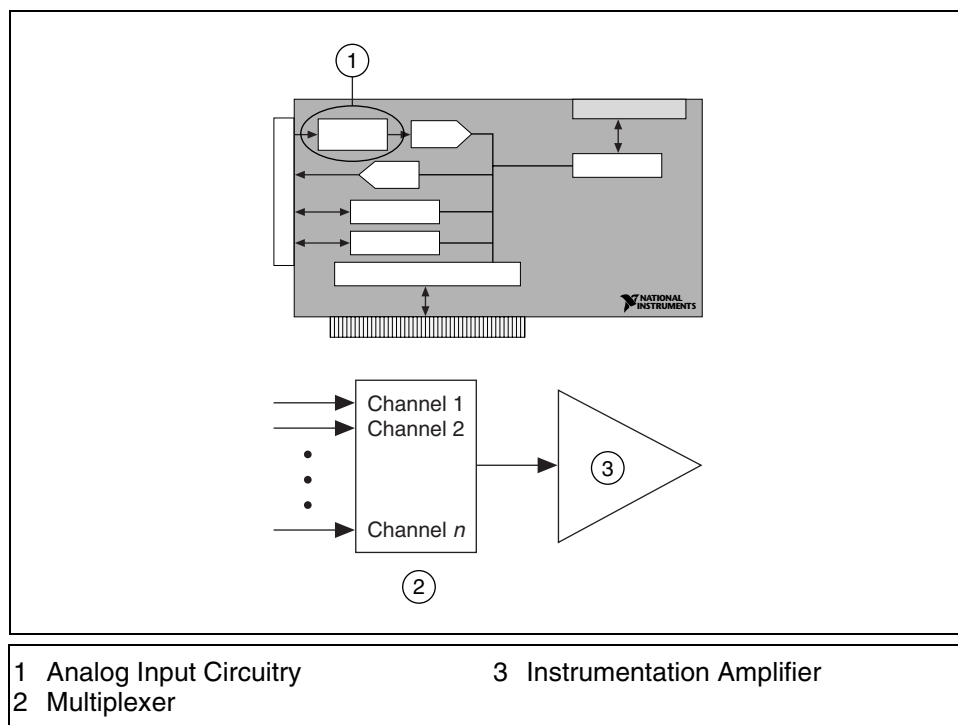
I/O Connector—The I/O connector is the means by which the signal enters or leaves the DAQ device. The I/O connector has 100-, 68-, or 50-pins, depending on the device. One end of the cable is connected to the I/O connector and the other end is connected to the terminal block. You can find the specific pinout in the DAQ device documentation.

Computer I/O Interface Circuitry—The computer I/O interface circuitry transfers information between the DAQ device and the computer. The computer I/O interface circuitry can differ depending on the bus protocol you use. For example, the PCI bus has connection leads that plug into a PCI slot, but the USB connection requires a cable.

RTSI Bus—The RTSI bus shares and synchronizes signals between multiple DAQ devices in the same computer. For example, if you want two devices to perform analog input at the same rate, you can share a clock signal over the RTSI bus so both devices use the same clock signal. Use a RTSI cable to connect the devices together unless you are using the PXI platform. The PXI chassis has a backplane that the devices plug into that acts as a built-in RTSI cable to share signals among any modules in the chassis.

Analog Input Circuitry

After entering the I/O connector, the analog input signal passes through the analog input circuitry before it passes to the analog-to-digital converter (ADC). The analog input circuitry consists of a multiplexer and an instrumentation amplifier. The following illustration shows details of the analog input circuitry.

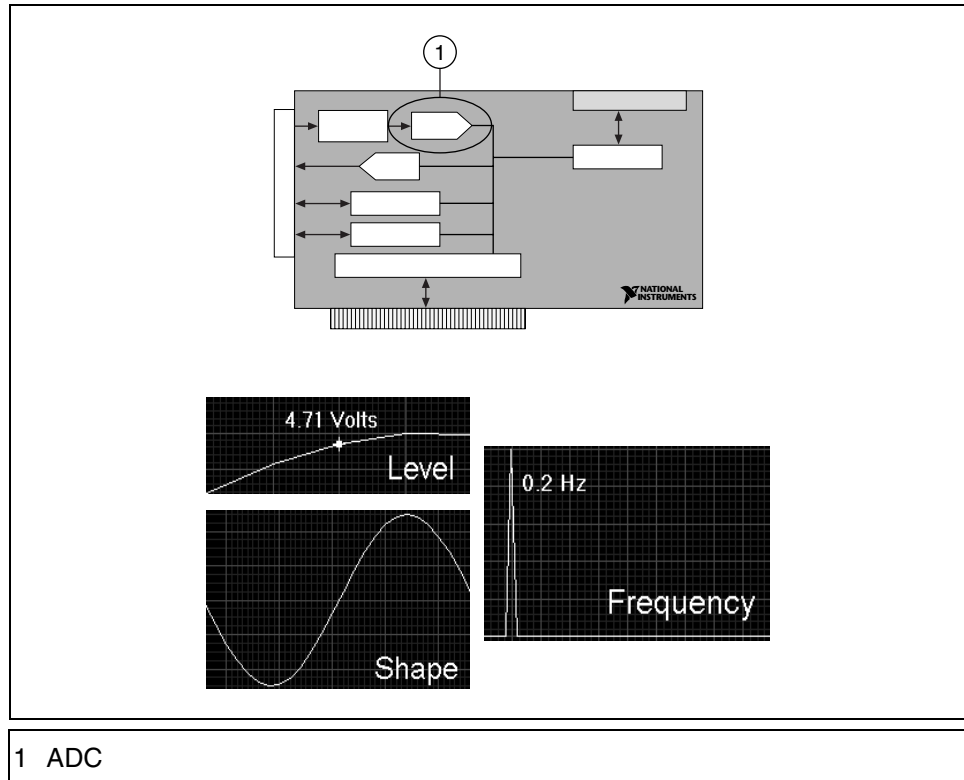


Multiplexer—The multiplexer, or mux, is a switch that connects only one of several input channels to the instrumentation amplifier at a time. When you acquire data from multiple channels, the mux rotates through the channels, connecting them one at a time to the amplifier. LabVIEW controls the order in which the mux connects the incoming signals to the amplifier.

Instrumentation Amplifier—The instrumentation amplifier can amplify or attenuate the signal that it receives. The purpose of the amplifier is to make the signal fill the range of the ADC as much as possible. When an amplifier amplifies or attenuates the signal, it is referred to as applying a gain. The gain is the amount of amplification that is applied. For example, a gain of two amplifies the signal by two. A gain of 0.5 makes the signal two times smaller, thus attenuating the signal.

Analog-to-Digital Converter (ADC)

The ADC converts an analog voltage into a digital number that you can send to the computer for interpretation using the computer I/O interface circuitry. The analog input circuitry combines with the ADC to acquire an analog signal so you can measure the level, shape, or frequency of that signal. You can use the analog input functionality of the DAQ device in applications ranging from testing a power supply to measuring a heartbeat to analyzing speech. The following illustration shows an ADC.

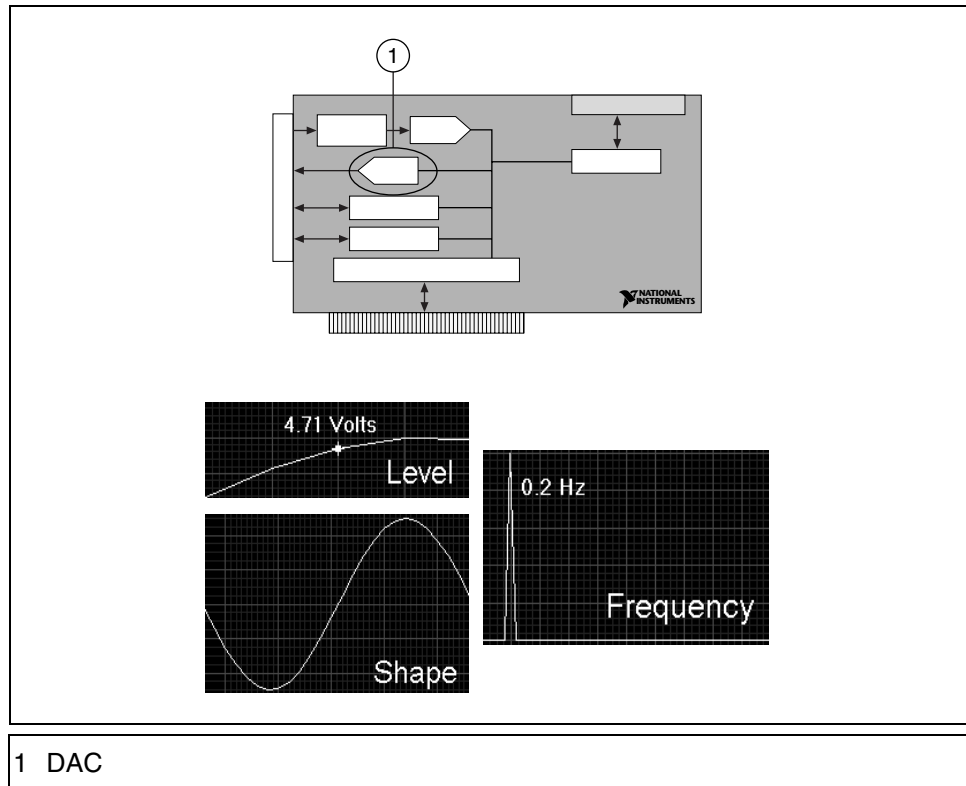


1 ADC

Refer to the *ADC* section of Appendix A of this manual for more information about ADCs.

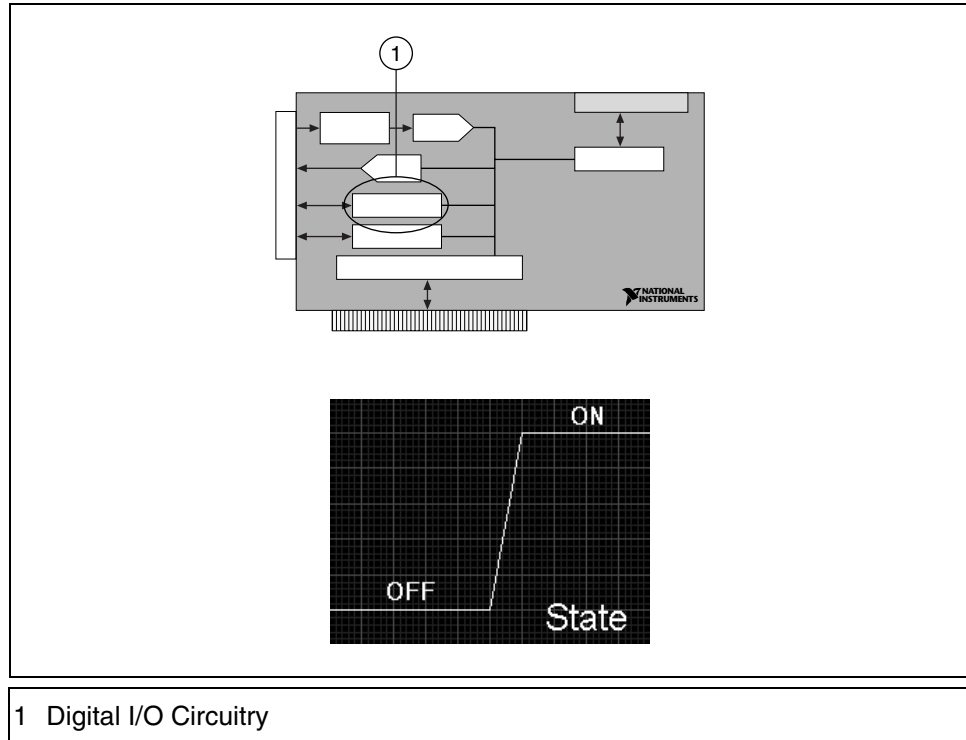
Digital-to-Analog Converter (DAC)

A DAC performs the opposite task of an ADC. It takes a digital number that was sent from the computer through the computer I/O interface circuitry and converts it into an analog signal that is output through the I/O connector. A DAC is useful for generating DC signals (level), specific tones (frequencies), and waveforms (shapes). You can use the analog output functionality of a DAQ device in applications ranging from control systems using a proportional integral derivative (PID) control, to controlling servo motors, to generating a series of specific tones for a siren or alarm. The following illustration shows a DAC.



Digital I/O Circuitry

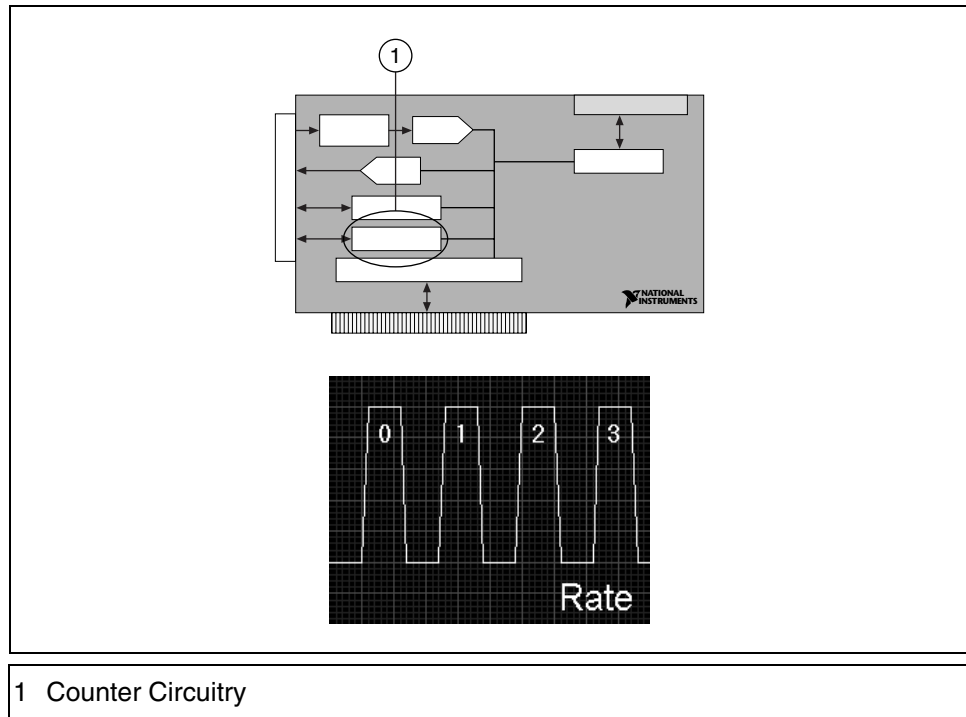
The digital I/O circuitry can perform both input and output functions. A typical E Series DAQ device has eight digital lines that can acquire or generate digital signals. The digital lines do not have timing or handshaking circuitry. Therefore, they are useful for measuring the state of a digital signal but not the rate. You can use the digital I/O functionality of the DAQ device in applications ranging from monitoring a switch to see if it has changed states to controlling a relay. The following illustration shows details of digital I/O circuitry.



1 Digital I/O Circuitry

Counter Circuitry

Counters acquire and generate digital signals. They have built-in timing signals called timebases that make them ideal for measuring the rate of a digital signal. You can use the counter functionality of a DAQ device in applications ranging from measuring the frequency of a motor shaft to controlling stepper motors by generating a specific frequency pulse train. The following illustration shows details of counter circuitry.



1 Counter Circuitry

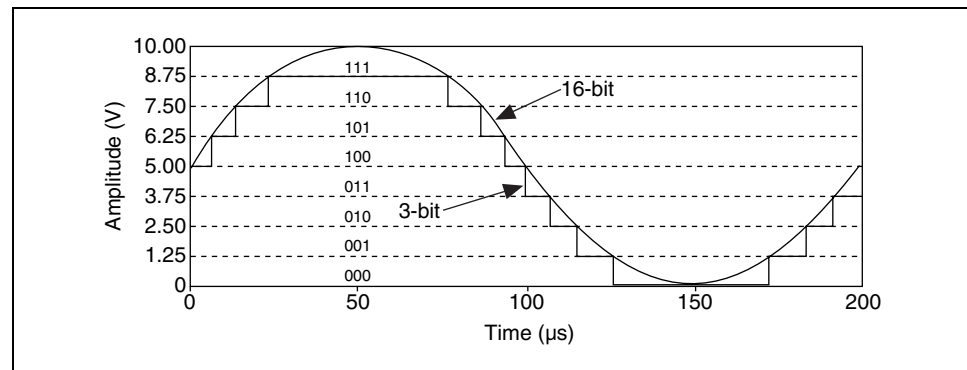
C. Configuration Considerations

There are several aspects of analog input and analog output circuitry that affect how you configure a DAQ device.

- The resolution and range of the ADC
- The gain applied by the instrumentation amplifier
- The combination of the resolution, range, and gain to calculate a property called the code width value

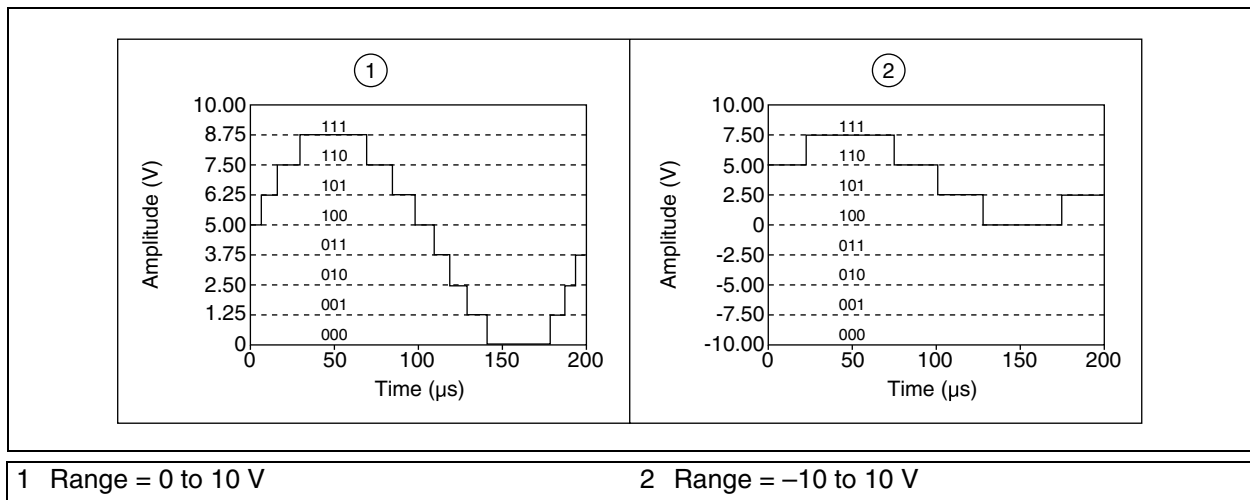
Resolution

The number of bits used to represent an analog signal determines the resolution of the ADC. The resolution on a DAQ device is similar to the marks on a ruler. The more marks a ruler has, the more precise the measurements are. The higher the resolution is on a DAQ device, the higher the number of divisions into which a system can break down the ADC range, and therefore, the smaller the detectable change. A 3-bit ADC divides the range into 2^3 or eight divisions. A binary or digital code between 000 and 111 represents each division. The ADC translates each measurement of the analog signal to one of the digital divisions. The following illustration shows a 5 kHz sine wave digital image obtained by a 3-bit ADC. The digital signal does not represent the original signal adequately because the converter has too few digital divisions to represent the varying voltages of the analog signal. However, increasing the resolution to 16 bits to increase the ADC number of divisions from eight (2^3) to 65,536 (2^{16}) allows the 16-bit ADC to obtain an extremely accurate representation of the analog signal.



Device Range

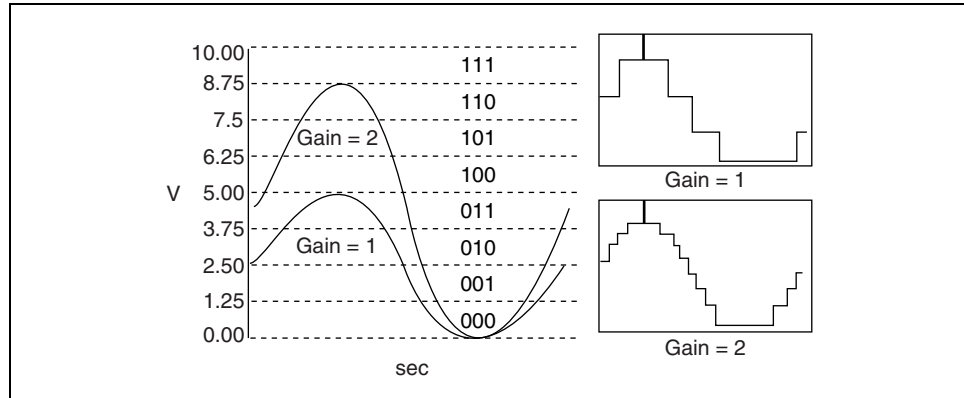
Range refers to the minimum and maximum analog signal levels that the ADC can digitize. Many DAQ devices feature selectable ranges (typically 0 to 10 V or -10 to 10 V), so you can match the ADC range to that of the signal to take best advantage of the available resolution to accurately measure the signal. For example, in the following illustration, the 3-bit ADC in chart 1 has eight digital divisions in the range from 0 to 10 V, which is a unipolar range. If you select a range of -10 to 10 V, which is a bipolar range, as shown in chart 2, the same ADC separates a 20 V range into eight divisions. The smallest detectable voltage increases from 1.25 to 2.50 V, and the right chart is a much less accurate representation of the signal.



Amplification

Amplification or attenuation of a signal might occur before the signal is digitized to improve the representation of the signal. By amplifying or attenuating a signal, you can effectively decrease the input range of an ADC and thus allow the ADC to use as many of the available digital divisions as possible to represent the signal.

For example, the following illustration shows the effects of applying amplification to a signal that fluctuates between 0 and 5 V using a 3-bit ADC and a range of 0 to 10 V. With no amplification, or gain = 1, the ADC uses only four of the eight divisions in the conversion. By amplifying the signal by two before digitizing, the ADC uses all eight digital divisions, and the digital representation is much more accurate. Effectively, the device has an allowable input range of 0 to 5 V because any signal above 5 V when amplified by a factor of two makes the input to the ADC greater than 10 V.



The range, resolution, and amplification available on a DAQ device determine the smallest detectable change in the input voltage. This change in voltage represents one least significant bit (LSB) of the digital value and is also called the code width.

Code Width

Code width is the smallest change in a signal that a system can detect. Code width is calculated using the following formula:

$$\text{code width} = \frac{\text{voltage range}}{(\text{amplification} \times 2^{\text{resolution in bits}})}$$

The smaller the code width, the more accurately a device can represent the signal. The formula confirms what you have already learned in the discussion on resolution, range, and gain:

- Larger resolution = smaller code width = more accurate representation of the signal
- Larger amplification = smaller code width = more accurate representation of the signal
- Larger range = larger code width = less accurate representation of the signal

Determining the code width is important in selecting a DAQ device. For example, a 12-bit DAQ device with a 0 to 10 V input range and an amplification of one detects a 2.4 mV change, while the same device with a -10 to 10 V input range would detect a change of 4.8 mV.

$$\frac{\text{range}}{\text{amplification} \times 2^{\text{resolution}}} = \frac{10}{1 \times 2^{12}} = 2.4 \text{ mV} \quad \frac{20}{1 \times 2^{12}} = 4.8 \text{ mV}$$

Exercise 2-1 Resolution, Range, Amplification, and Code Width

Objective: To determine the optimal configuration for a data acquisition measurement system.

In this exercise, you will review the specifications for three projects and determine the most appropriate DAQ device and the correct configuration settings for that device for each project.

When choosing a DAQ device, consider performance against cost. A higher resolution DAQ device costs more but provides a more accurate representation of the acquired signal. A DAQ device with a larger range provides more flexibility but adversely affects the code width. The following table lists four DAQ device configurations you can use in each of the projects in this exercise. Typical DAQ devices have either 12- or 16-bit resolution and a choice of bipolar or unipolar inputs.

The DAQ device information in the following table represents typical DAQ devices. In each project of this exercise, select the best DAQ device and the optimum configuration to maximize cost and accuracy. The best approach is to first determine whether you need a bipolar or unipolar range and determine the appropriate level of amplification. Then determine if the desired code width is within the capability of the 12- or 16-bit DAQ device.



Tip Use the code width equation.

$$\text{code width} = \frac{\text{voltage range}}{(\text{amplification} \times 2^{\text{resolution in bits}})}$$

The following table lists the two DAQ devices and various configurations:

	DAQ Device 1		DAQ Device 2	
	Config. A	Config. B	Config. A	Config. B
Resolution (Bits)	12 Bit	12 Bit	16 Bit	16 Bit
Range (Volts)	0 to 10	-10 to 10	0 to 10	-10 to 10
Amplification (choose one)	1, 2, 5, 10, 20, 50, 100			

Project 1

A K-type thermocouple is attached to the steam drum output of a high-pressure boiler system. The thermocouple can measure a temperature range of -270 to $1,372$ °C. With this temperature range, the thermocouple returns a voltage of -6.548 to 54.874 mV.



Note Do not expect the boiler ever to freeze.

To detect a change of 2.1 μ V, what is the best DAQ device and configuration for the project?

Project 2

A pressure transducer is placed on the intake manifold of an engine. The transducer outputs a voltage between -2 V and 2 V for a linear pressure range of 20 Pa to 105 kPa.

To detect a change of 1.5 Pa, which would be a voltage change of 70 μ V, what is the best DAQ device and configuration for the project?

Project 3

A humidity sensor is placed in a weather station to monitor relative humidity. The sensor outputs a linear voltage between 0.8 V and 3.9 V for a relative humidity of 0% to 100% . The sensor is accurate between $\pm 2\%$ relative humidity.

To detect a voltage change of at least 0.62 mV, what is the best DAQ device and configuration for the project?

Solutions

Project 1—In this project, because you can assume the boiler will not freeze, the thermocouple always produces a positive voltage. Therefore, you can select DAQ Device 1 Configuration A or DAQ Device 2 Configuration A because these configurations are unipolar. Since you are using a thermocouple that has a maximum voltage of 54.874 mV, you can apply a maximum amplification to the signal. So, you can assume an amplification of 100. Your only decision is whether or not you need a 12- or 16-bit device. Using the code width equation, if you select DAQ Device 1 Configuration A with an amplification of 100, the code width is 24 μV , so this DAQ device cannot detect a change of 2.1 μV . Using the same equation for DAQ Device 2 Configuration A, the code width is 1.5 μV . Therefore, DAQ Device 2 running in Configuration A is the best DAQ device for this project.

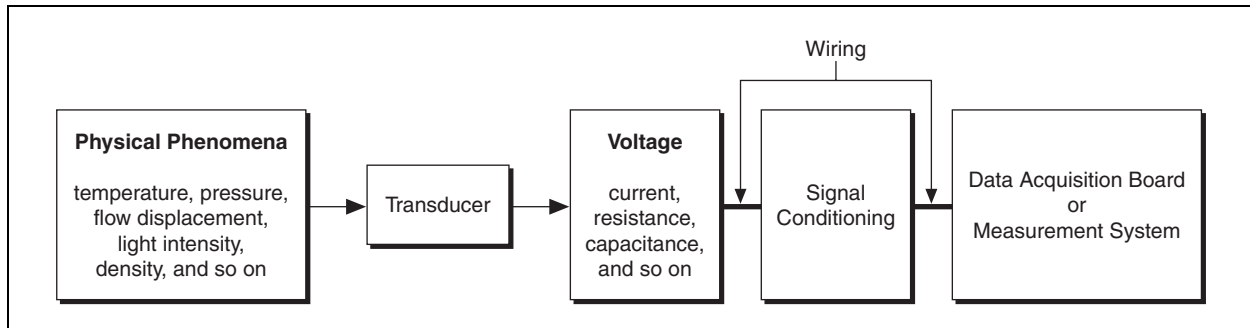
Project 2—In this project, the transducer is linear and outputs a bipolar voltage. Therefore, you need to select DAQ Device 1 Configuration B or DAQ Device 2 Configuration B. If you apply an amplification of 5 to the signal, you will be able to use the entire range of the DAQ device. Using the code width equation, DAQ Device 1 Configuration B can detect a change of 977 μV . DAQ Device 2 Configuration B can detect a change of 61 μV . So, DAQ Device 2 Configuration B is best able to detect a change of 70 μV . Therefore, DAQ Device 2 Configuration B is the best DAQ device for this project.

Project 3—In this project, the humidity sensor generates a linear voltage with respect to the relative humidity. Because the sensor is unipolar, you can use DAQ Device 1 Configuration A or DAQ Device 2 Configuration A. The maximum voltage the sensor can output is 3.9 V, so you can select an amplification of 2. If you use DAQ Device 1 Configuration A, the code width of the device is 1.2 mV. The code width of DAQ Device 2 Configuration A is 76 μV . Because the device for this project needs to detect a voltage change of 0.62 mV, DAQ Device 2 running in Configuration A is the best device for this project.

End of Exercise 2-1

D. Grounding Issues

To produce accurate and noise-free measurements, knowledge of the nature of the signal source, a suitable configuration of the DAQ device, and an appropriate cabling scheme might be required. The following illustration shows a typical data acquisition system. The integrity of the acquired data depends on the entire analog signal path.



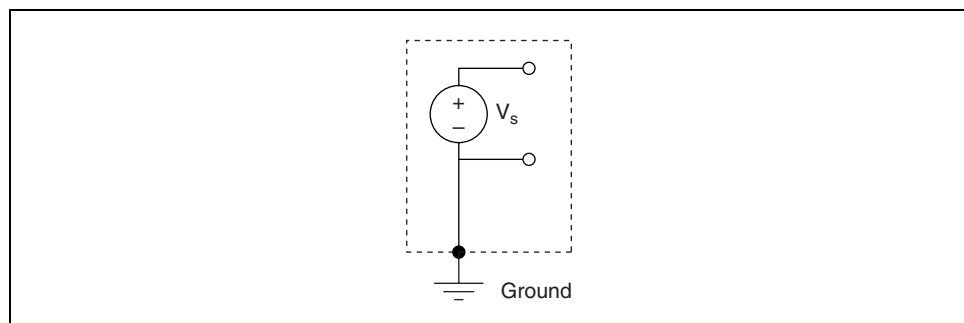
In order to cover a wide variety of applications, most DAQ devices provide some flexibility in their analog input stage configuration. The price of this flexibility can lead to confusion about the proper applications of the various input configurations and their relative merits. The purpose of this part of the course is to help clarify the types of input configurations available on DAQ devices and to explain how to choose and use the configuration best suited for the application. An understanding of the types of signal sources and measurement systems is a prerequisite to apply good measurement techniques.

Types of Signal Sources

Analog input acquisitions use grounded and floating signal sources.

Grounded Signal Sources

A grounded 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 the following illustration.



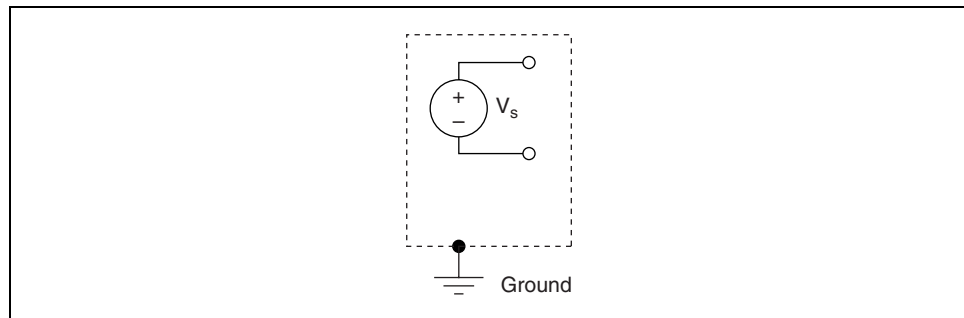
Because such sources use the system ground, they share a common ground with the measurement device. The most common examples of grounded sources are devices that plug into a building ground through wall outlets, such as signal generators and power supplies.



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.

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 the following illustration.



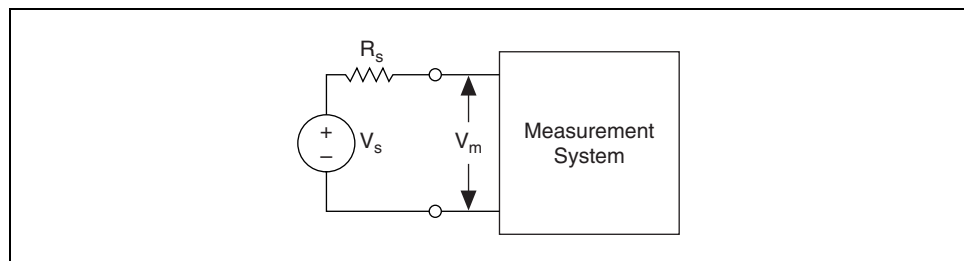
Some common examples of floating signal sources are batteries, thermocouples, transformers, and isolation amplifiers. Notice in the illustration of the floating signal source that neither terminal of the source is connected to the electrical outlet ground as in the previous illustration of a grounded signal source. Each terminal is independent of the system ground.

E. Types of Measurement Systems

The most common electrical equivalent produced by signal conditioning circuitry associated with transducers is in the form of voltage.

Transformation to other electrical phenomena such as current and frequency may be encountered in cases where the signal is to be carried over long cable distances in harsh environments. Because in virtually all cases the transformed signal is ultimately converted back into a voltage signal before measurement, it is important to understand the voltage signal source.

A voltage signal is measured as the potential difference across two points, as shown in the following illustration.

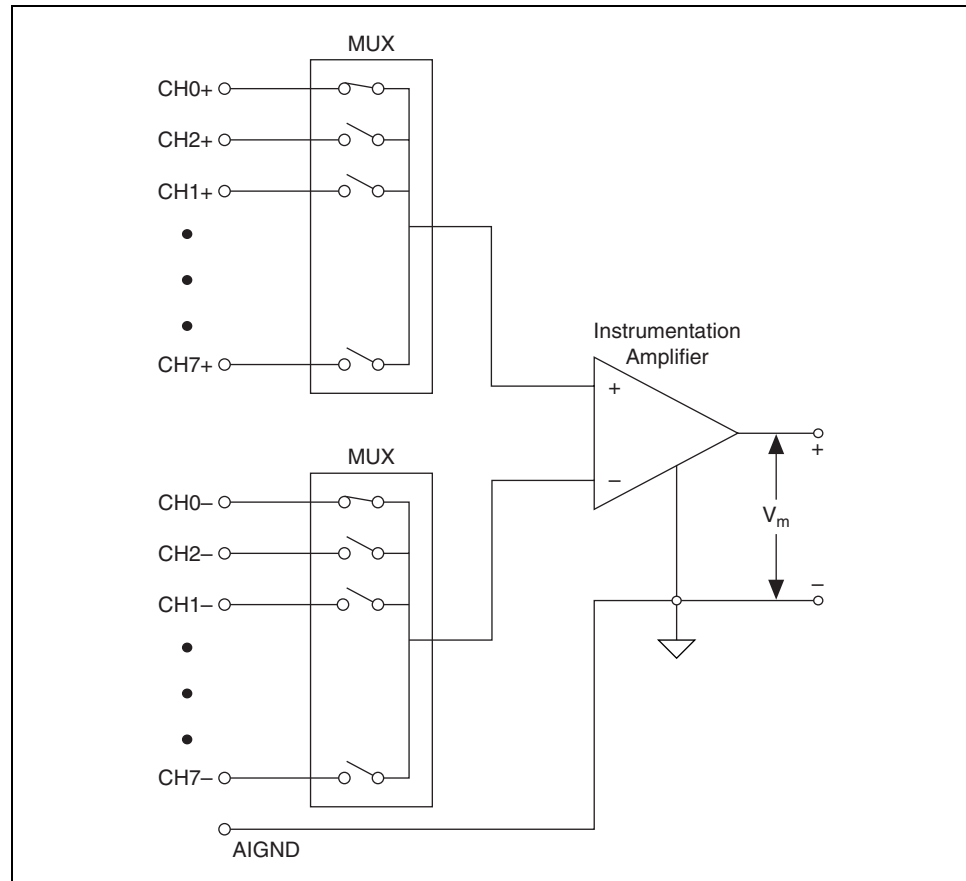


You configure a measurement system based on the hardware you use and the measurement you take.

Differential Measurement System

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 system as shown in the following illustration. Using analog multiplexers in the signal path increases the number of measurement channels when only one instrumentation amplifier exists. In the following illustration, the AIGND (analog input ground) pin is the measurement system ground.



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 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}):

$$V_{cm} = (V_+ + V_-) / 2$$

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

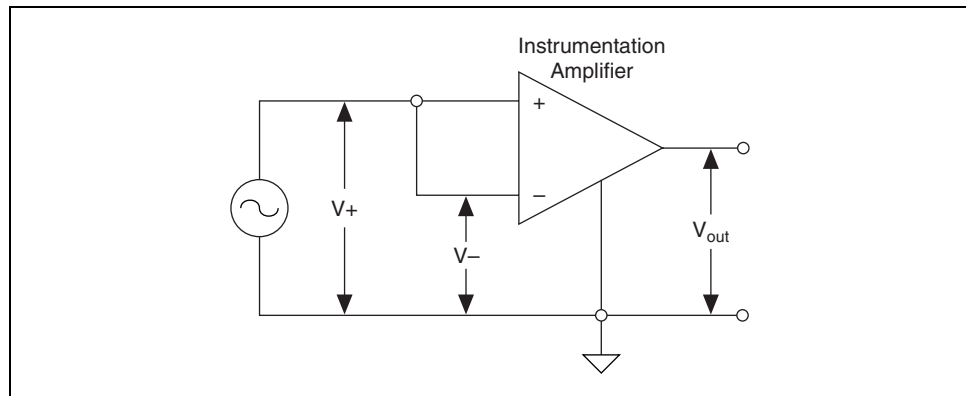
CMRR measures the ability of a differential measurement system to reject the common-mode voltage signal. The CMRR is a function of frequency and typically reduces with frequency. The higher the CMRR, the better the amplifier can extract differential signals in the presence of common-mode noise. Using 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):

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

The following illustration shows a simple circuit in which CMRR in dB is measured as:

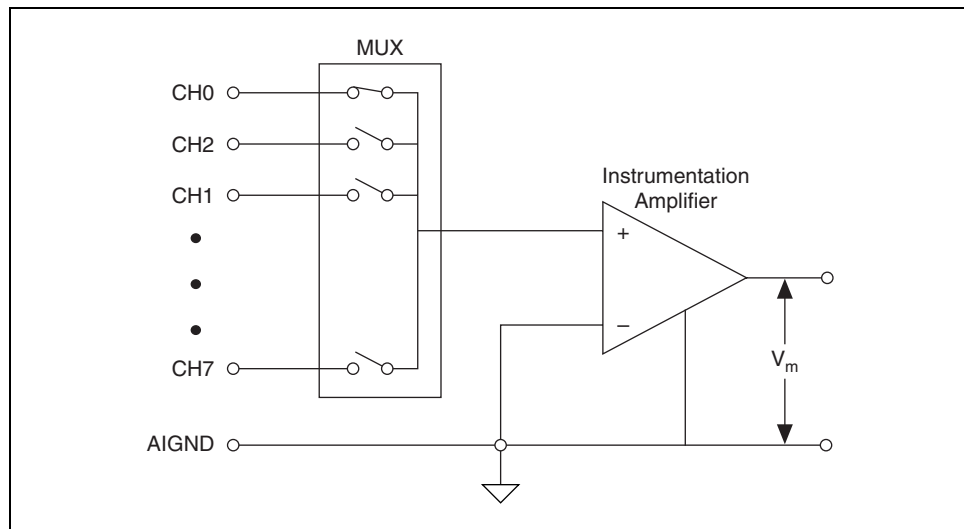
$$20 \log \frac{V_{cm}}{V_{out}}$$

where $V_+ + V_- = V_{cm}$

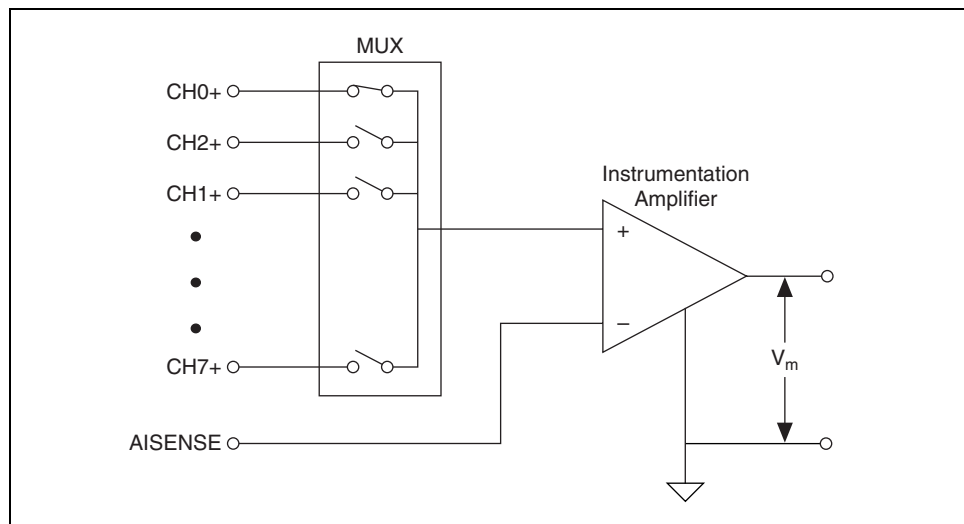


Referenced and Non-Referenced Single-Ended Measurement Systems

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. The following illustration shows an eight-channel 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. The following illustration shows a NRSE system.



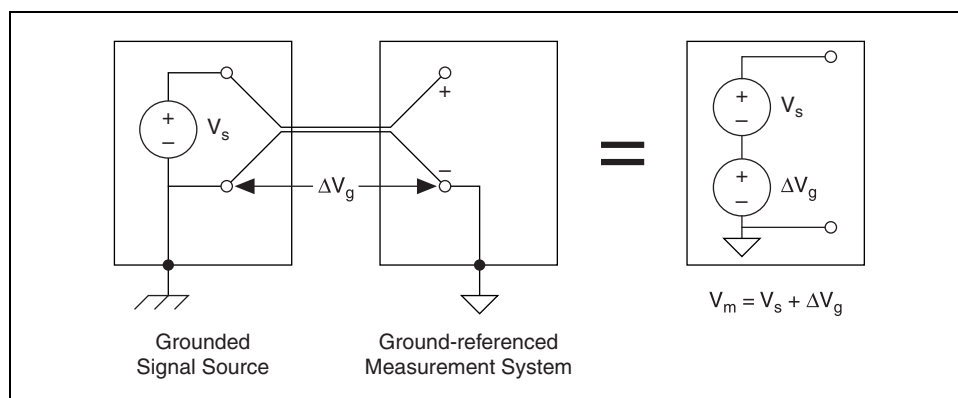
In a NRSE measurement system, all measurements are still made with respect to a single-node analog input sense (AISENSE on E Series devices), but the potential at this node can vary with respect to the measurement system ground (AIGND). A single-channel NRSE measurement system is the same as a single-channel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

F. Measuring Signal Sources

There are two types of signal sources—grounded signal sources and floating non-referenced signal sources.

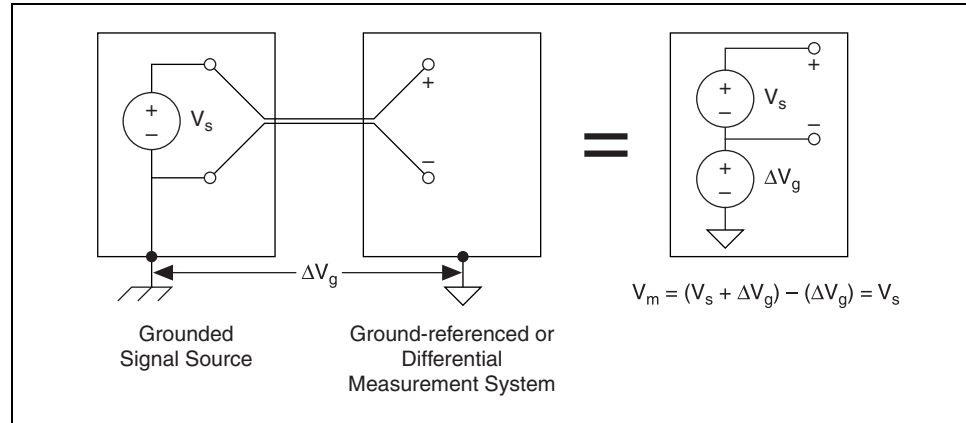
Measuring Grounded Signal Sources

A grounded signal source is best measured with a differential measurement system. The following illustration shows the pitfall of using a ground-referenced measurement system to measure a grounded signal source. In this case, the measured voltage, V_m , is the sum of the signal voltage, V_s , and the potential difference, ΔV_g , that exists between the signal source ground and the measurement system ground. This potential difference is generally not a DC level. A noisy measurement system results, often showing power-line frequency (60 Hz) components in the readings. Ground-loop introduced noise may have both AC and DC components, introducing offset errors and noise in the measurements. The potential difference between the two grounds causes a ground-loop current to flow in the interconnection.



You can still use a ground-referenced system if the signal voltage levels are high and the interconnection wiring between the source and the measurement device has a low impedance. In this case, the signal voltage measurement is degraded by ground loop, but the degradation may be tolerable. You must observe the polarity of a grounded signal source before connecting it to a ground-referenced measurement system because the signal source might be shorted to ground, which can damage the signal source.

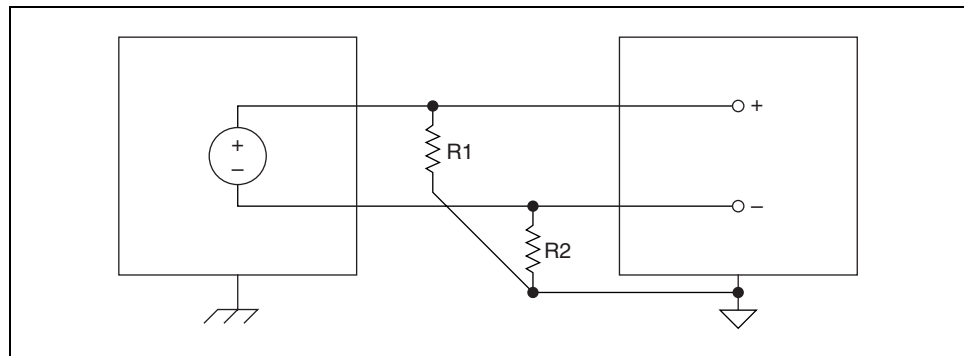
Both the differential (DIFF) and the NRSE input configurations provide non-referenced measurements on a typical DAQ device. With either configuration, differences between references of the source and the measuring device appear as common-mode voltage to the measurement system, and are subtracted from the measured signal. The following illustration illustrates this concept.



Measuring Floating Sources

You can measure floating signal sources with both differential and single-ended measurement systems. In the case of the differential measurement system, however, make sure the common-mode voltage level of the signal with respect to the measurement system ground remains in the common-mode input range of the measurement device.

A variety of phenomena—for example, the instrumentation amplifier input bias currents—can move the voltage level of the floating source out of the valid range of the input stage of a DAQ device. To anchor this voltage level to some reference, use resistors as shown in the following illustration.



These bias resistors provide a DC path from the instrumentation amplifier inputs to the instrumentation amplifier ground. If you do not use resistors and the source is truly floating, the source is not likely to remain within the common-mode signal range of the instrumentation amplifier, and the amplifier saturates. You must reference the source to AIGND. The easiest way to connect the signal source to the measurement system is to connect the positive side of the signal to the positive input of the instrumentation amplifier and connect the negative side of the signal to AIGND and to the negative input of the instrumentation amplifier without bias resistors. This connection works well for DC-coupled sources with a source impedance less than 100 Ω . For larger source impedances, this connection leaves the

signal path significantly out of balance. Noise that couples electrostatically onto the positive line does not couple onto the negative line because it is connected to AIGND. Therefore, this noise appears as a differential-mode signal instead of a common-mode signal, and the instrumentation amplifier does not reject it. In this case, instead of directly connecting the negative line to AIGND, connect it to AIGND through a resistor that is about 100 times the equivalent source impedance. The resistor puts the signal path nearly in balance, so that about the same amount of noise couples onto both connections, yielding better rejection of electrostatically coupled noise.

You can fully balance the signal path by connecting another resistor of the same value between the positive input and AIGND as shown in the previous illustration. This fully balanced configuration offers slightly better noise rejection but has the disadvantage of loading down the source with the series combination (sum) of the two resistors. If, for example, the source impedance is 2 k Ω and each of the two resistors is 100 k Ω , the resistors load down the source with 200 k Ω and produce a -1% gain error.

Both inputs of the instrumentation amplifier require a DC path to ground for the instrumentation amplifier to work. If the source is AC coupled (capacitively coupled), the instrumentation amplifier needs a resistor between the positive input and AIGND. If the source has low impedance, choose a resistor that is large enough not to significantly load the source but small enough not to produce significant input offset voltage as a result of input bias current (typically 100 k Ω to 1 M Ω). In this case, you can tie the negative input directly to AIGND. If the source has high output impedance, you should balance the signal path as previously described using the same value resistor on both the positive and negative inputs. Some gain error results from loading down the source.

Resistors provide a return path to ground for instrumentation amplifier input bias currents. Only R2 is required for DC-coupled signal sources. For AC-coupled sources, R1 = R2.



Caution Failure to use these resistors results in erratic or saturated (positive full-scale or negative full-scale) readings.

If you use single-ended input mode, you can use an RSE input system for a floating signal source. No ground loop is created in this case. You also can use the NRSE input system, which is preferable from a noise pickup point of view. Floating sources require bias resistor(s) between the AISENSE input and the measurement system ground (AIGND) in the NRSE input configuration.

Summary of Signal Sources and Measurement Systems

The following illustration 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 <ul style="list-style-type: none"> • Ungrounded Thermocouples • Signal Conditioning with Isolated Outputs • Battery Devices 	Examples <ul style="list-style-type: none"> • Plug-in Instruments with Nonisolated Outputs
Differential (DIFF)		
Single-Ended — Ground Referenced (RSE)		<p style="text-align: center;">NOT RECOMMENDED</p> <p style="text-align: center;">Ground-loop losses, V_g, are added to measured signal.</p>
Single-Ended — Nonreferenced (NRSE)		

Exercise 2-2 Differential, RSE, and NRSE

Objective: To learn how to choose a grounding mode for a measurement system and how to properly connect signals to that measurement system.

Part I

Assume you have an instrument that plugs into a standard wall outlet. The outputs of the instrument are referenced to the same ground as the instrument. Which measurement system would you use when connecting the outputs of the instrument to a DAQ device in the computer? (circle one)

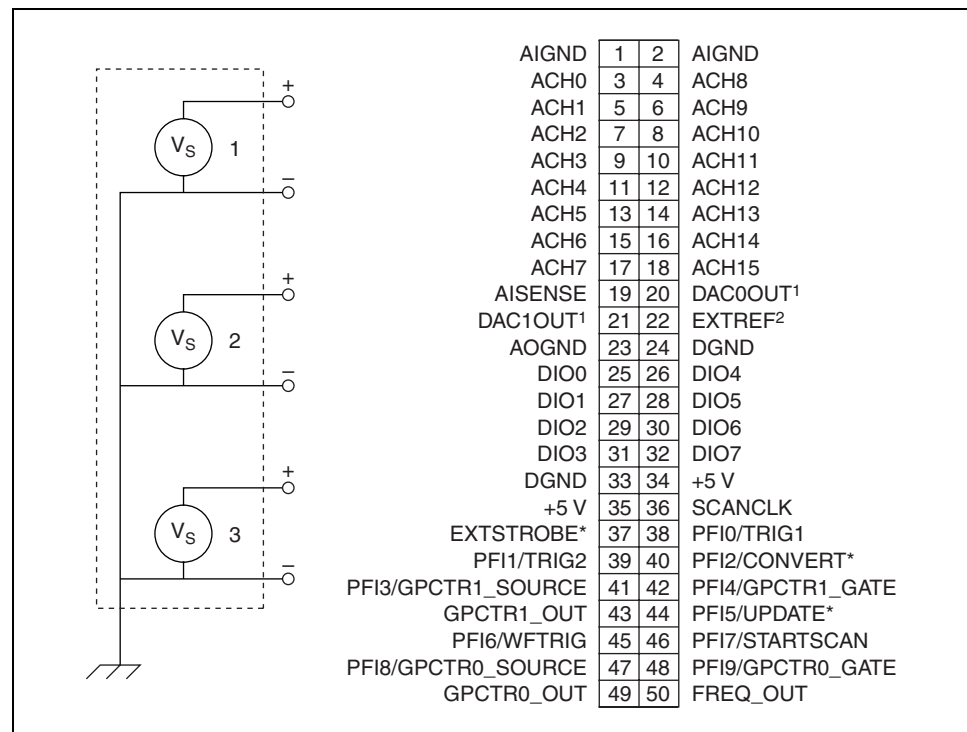
Differential

RSE

NRSE

The following illustration shows the instrument and a 50-pin layout for a PCI-MIO-16E-4. Based on your choice for the grounding mode of the measurement system, connect the following:

- voltage source 1 to analog input channel 0
- voltage source 2 to analog input channel 1
- voltage source 3 to analog input channel 2



Solution to Part I

The signal source is grounded, so you cannot choose RSE. The ideal choice is differential because you do not have more than eight signals to measure. If you have more than eight signals to measure, the ideal choice is NRSE. If you chose differential, wire the positive lead from voltage source 1 to pin 3 and wire the negative lead to pin 4. Wire the positive lead from voltage source 2 to pin 5 and wire the negative lead to pin 6. Finally, wire the positive lead from voltage source 3 to pin 7 and wire the negative lead to pin 8. If you chose NRSE, wire the positive leads in the same way, but wire the negative leads to AISENSE.

Part II

Assume you have three batteries. Which measurement system would you use when connecting the outputs of the batteries to a DAQ device in the computer? (circle one)

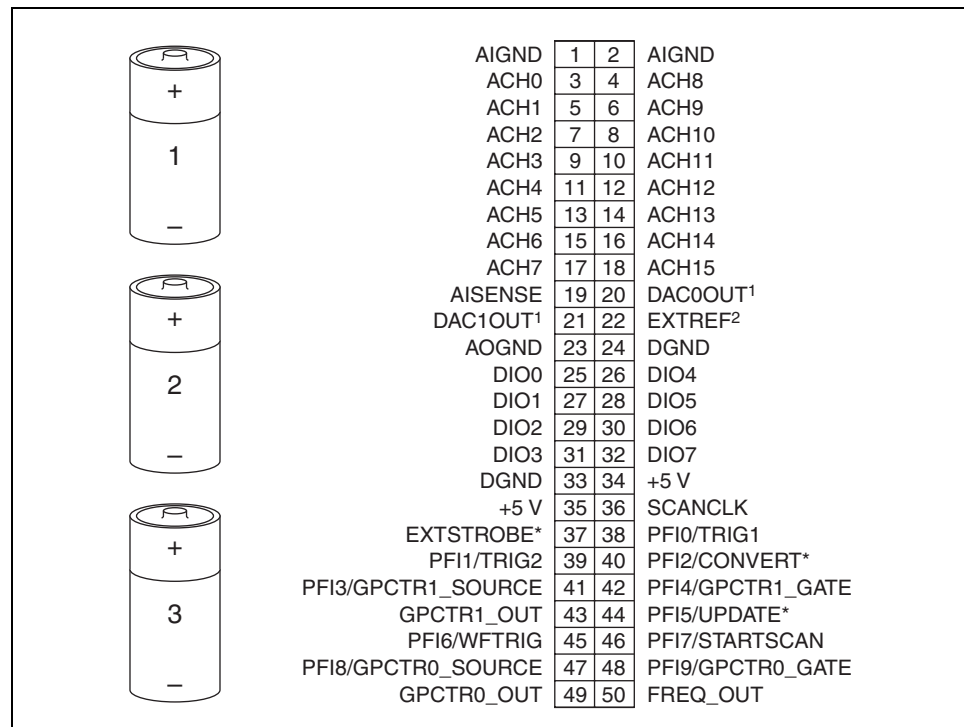
Differential

RSE

NRSE

The following illustration shows the batteries and a 50-pin layout for a PCI-MIO-16E-4. Based on your choice for the grounding mode of the measurement system, connect the following:

- battery 1 to analog input channel 5
- battery 2 to analog input channel 6
- battery 3 to analog input channel 7



Solution to Part II

The signal source is floating, so you could choose any of the three measurement systems. Both RSE and NRSE will allow you to use 16 channels, but NRSE requires you to use bias resistors, so eliminate NRSE. Because you have fewer than eight signals to measure, the best choice for a good measurement is differential. However, with differential you would need bias resistors, so the simplest choice is RSE. Assuming you chose differential, wire the positive lead from battery 1 to pin 13 and wire the negative lead to pin 14. Wire the positive lead from battery 2 to pin 15 and wire the negative lead to pin 16. Finally, wire the positive lead from battery 3 to pin 17 and wire the negative lead to pin 18. You also need bias resistors from the negative terminal of each battery to AIGND. If you chose RSE, wire the positive leads in the same way, but wire all the negative terminals to AIGND. RSE does not require bias resistors.

End of Exercise 2-2

G. DAQ Software

The final component of a complete DAQ system is the 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.

A measurement system consists of the following software applications:

- NI-DAQ—Software that controls the DAQ device.
- Measurement & Automation Explorer (MAX)—Software that communicates between LabVIEW and NI-DAQ.
- LabVIEW—Software that you use to create an application to send commands to the driver and acquire, analyze, and present data.

H. 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. You use NI-DAQ software to communicate with National Instruments DAQ devices such as E Series multifunction I/O (MIO) devices, SCXI signal conditioning modules, and switch modules. This course describes LabVIEW development using only NI-DAQmx.



Note You cannot use NI-DAQ with third-party DAQ devices. You might need to contact the vendor of the third-party device to obtain a driver specific to that device.

NI-DAQ is compatible with the following software applications and programming languages:

- LabVIEW
- Measurement Studio
- Microsoft Visual C/C++
- Microsoft .NET Languages
- Visual Basic
- ANSI C

Refer to the *NI-DAQ 7.0 Readme* for more information about the compatibility of NI-DAQ with specific versions of these languages.

Traditional NI-DAQ

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

NI-DAQmx is the latest NI-DAQ driver with new VIs, functions, and development tools for controlling measurement devices. NI-DAQmx provides a user interface and set of tools for programming and configuring your DAQ device. NI-DAQmx includes the following advantages over previous versions of NI-DAQ:

- The DAQ Assistant, 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 NI-DAQmx code to another DAQ system. Use LabVIEW or MAX to launch the DAQ Assistant.

- Increased performance, including faster single-point analog I/O and multithreading.
- A simpler API for creating DAQ applications using fewer functions and VIs than earlier versions of NI-DAQ.
- Expanded functionality in LabVIEW including Property Nodes for data acquisition and improved waveform data type support for both analog and digital I/O.
- Similar API and functionality for ANSI C, LabWindows™/CVI™, and Measurement Studio, including native .NET and C++ interfaces.

Traditional NI-DAQ and NI-DAQmx support different sets of devices. Refer to the *NI-DAQ 7.0 Readme* or the *DAQ Quick Start Guide* for lists of supported devices.

I. Measurement & Automation Explorer

Measurement & Automation Explorer (MAX) is a Windows-based application installed during NI-DAQ installation. Use MAX to configure and test NI software and hardware, 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. MAX includes the following function categories:

- Data Neighborhood
- Devices and Interfaces
- IVI Instruments
- Scales
- Historical Data
- Software
- VI Logger Tasks

Data Neighborhood

Data Neighborhood provides access to descriptively named shortcuts to configure physical channels in your system, including DAQ virtual channels and tasks. The Data Neighborhood category also provides utilities for testing and reconfiguring those virtual channels. You also can access the DAQ Assistant from Data Neighborhood to create and configure settings for virtual channels and tasks.

DAQ Assistant

The DAQ Assistant is a graphical interface for building your measurement channels and tasks.

- **Channel**—An NI-DAQmx channel maps configuration information such as scaling and input limits to a specified physical channel. You can set up the configuration information for the channel and give the channel a descriptive name at the same time. Later, you can use the descriptive name to access that channel and its configuration in a task or LabVIEW. You can give the channel a description, decide what type of transducer the channel will use, set the range (determines gain), choose the grounding mode, assign custom scaling for the virtual channel, and give the channel a descriptive name to replace the channel number all at the same time.

For example, assume channel 0 on the DAQ Signal Accessory is hardwired to a temperature sensor. You can create a virtual channel for channel 0 and call it Temperature Sensor. You can create virtual channels for analog I/O, counter I/O, and digital I/O. In this case,

referring to a channel by a name (Temperature Sensor) instead of a number (0) helps you remember what the channel does.

- **Task**—An NI-DAQmx task is a collection of channels with the same timing and triggering. A task represents a measurement or a generation that you wish to perform. The channels that compose the task can be used in multiple tasks (global channel) or assigned to just one specific task (local channel). You also can create new channels while creating a task or you can compose a task with channels that have been created using the DAQ Assistant.

Devices and Interfaces

The Devices and Interfaces category lists installed and detected NI hardware. The Devices and Interfaces category also includes the Self-Test, Test Panels, Reset, Properties, and Self-Calibrate utilities for configuring and testing devices.

Self-Test

The Self-Test utility runs an internal test on a DAQ device to ensure that all resources are properly assigned and that the device is configured correctly.

Test Panels

The Test Panel utility tests the analog I/O, digital I/O, and counter I/O functionality of a DAQ device. Use the Test Panel utility to troubleshoot the functionality of the device and system configuration directly from NI-DAQmx. If the device does not work in the Test Panel, it will not work in LabVIEW. If you experience problems with data acquisition in a LabVIEW program, run the Self-Test and Test Panel utilities to begin troubleshooting.

Reset

The Reset utility resets the DAQ device to its default state.

Properties

The Properties utility allows you to configure and view the RTSI configuration and accessory devices that are used with your DAQ device. The system resources for the device, such as the memory range and IRQ level, are listed in the **Attributes** tab in the window to the right of the Configuration window in MAX.

Self-Calibrate

The Self-Calibrate utility performs an internal calibration of the DAQ device.

Scales

The Scales category lists all the currently configured custom scales and provides utilities for testing and reconfiguring those custom scales. Scales also provides access to the DAQ Assistant, which allows you to create new custom scales.

DAQ Assistant

Use the DAQ Assistant to create custom scales you can use to determine scaling information for existing virtual channels. Each custom scale can have its own name and description to help you identify it. A custom scale can be one of the following four types:

- **Linear**—Scales that use the formula $y = mx + b$.
- **Map Ranges**—Scales in which values are scaled proportionally from a range of raw values to a range of scaled values.
- **Polynomial**—Scales that use the formula $y = a_0 + (a_1 * x) + (a_2 * x^2) + \dots + (a_n * x^n)$.
- **Table**—Scales in which you enter the raw value and the corresponding scaled value in a table format.

Software

The Software category shows all the currently installed NI software. The icon for each software package is also a shortcut that you can use to launch the software. The Software category also includes a Software Update Agent. The purpose of the Software Update Agent is to check if the National Instruments software is the latest version. If the software is not the latest version, the Software Update Agent opens the Web page on ni.com to download the latest version of the software.

Software Architecture for Windows

The main component of NI-DAQmx, the `nidaq32.dll`, makes function calls directly to a DAQ device. The function that the `nidaq32.dll` performs depends on where you access it from. Both MAX and LabVIEW can talk to NI-DAQmx. MAX is used primarily for configuring and testing the DAQ device. MAX not only helps configure devices, but it also tells you what devices are present in the system. To do this, MAX must communicate with the Windows Device Manager and the Windows Registry.

Exercise 2-3 Using Measurement & Automation Explorer

Objective: To become familiar with the **Devices and Interfaces** section of MAX and to explore the test panel functionality.

1. Connect the sine wave from the function generator to analog in 1 on the DAQ Signal Accessory.
2. Connect analog out 0 to analog in 2.



Note Do not alter any of the setup parameters during this exercise. The hardware has been correctly configured for the course, and any changes you make might cause errors during subsequent exercises.



3. Launch MAX by double-clicking the icon on the desktop or selecting **Start»Programs»National Instruments»Measurement & Automation Explorer**.
4. Double-click the **Devices and Interfaces** category. MAX searches for installed hardware and lists the devices found.
5. Double-click the **NI-DAQmx Devices** category. This listing indicates the type of DAQ device installed on your computer and the device name. The default value for the device name is "Dev1". If the device is not listed, select **View»Refresh**.
6. Right-click the DAQ device and select **Self-Test** from the shortcut menu. A dialog box appears that indicates the device has passed the test. Click the **OK** button to close the dialog box.



Note If the DAQ device does not pass the self-test, notify your instructor.

7. Right-click the DAQ device and select **Properties** from the shortcut menu. The **Device Properties** dialog box appears.

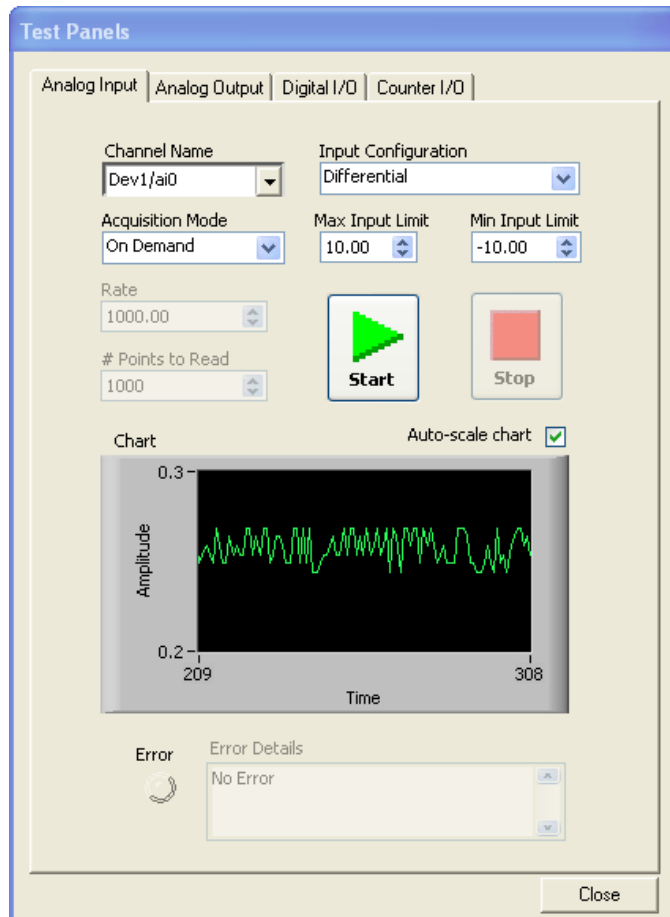
The **RTSI Configuration** tab specifies if a RTSI cable is attached to the device. RTSI cables route internal signals from one device to another.

The **Accessory** tab allows you to select and configure any accessories that are attached to the DAQ device. For example, if you use the SCB-68 terminal block to connect signals to the DAQ device, select it here. Devices on the Accessory list usually provide some form of signal conditioning for signals, or they increase the number of channels you can measure. If an accessory does not change the way signals are measured, it does not appear on the Accessory list. To use the DAQ Signal Accessory, leave **None** selected. The DAQ Signal Accessory is not included in the list because it does not change the measurement of the signals.

Click the **OK** button to exit the **Device Properties** dialog box.

8. Right-click the DAQ device and select **Test Panels** from the shortcut menu. The **Test Panels** dialog box appears. The **Analog Input** tab allows you to read the analog input channels.

Set the **Channel Name** to $\text{DevX}/\text{ai}0$, where X corresponds to the device number of your DAQ device. Channel 0 is the temperature sensor on the DAQ Signal Accessory. Click the **Start** button. A voltage between 0.2 and 0.3 V should display on the graph, as shown in the following figure.



9. Place your finger on the temperature sensor on the DAQ Signal Accessory to increase the voltage. Click the **Stop** button.



Note The test panel is an effective method of troubleshooting because it functions on a level closer to the hardware than LabVIEW. If the test panel functions properly and the LabVIEW VI does not work, the problem is with the LabVIEW VI. If the test panel does not work, the problem is with either the hardware or the driver configuration.

10. On the **Analog Input** test panel, change the channel to $\text{DevX}/\text{ai}1$, where X corresponds to the device number of your DAQ device. Make sure the sine wave from the function generator is connected to analog

in 1 on the DAQ Signal Accessory. Click the **Start** button. The sine wave might look distorted. Click the **Stop** button.

The **Acquisition Mode** pull-down menu includes the following options.

- **On Demand**—Acquires a single point of data.
- **Finite**—Displays only one screen of data.
- **Continuous**—Continuously displays a screen of data at a time.

The **Finite** and **Continuous** acquisition modes allow you to adjust the sample rate. The higher the sample rate, the more accurately the graph displays the waveform.

11. Complete the following steps to improve the accuracy of the graph display.
 - a. On the DAQ Signal Accessory, set the **frequency range** to 100 Hz – 10 kHz and turn the **Frequency Adjust** to **Lo**.
 - b. On the **Test Panel**, set the **Acquisition Mode** to **Finite** or **Continuous** and try different values such as 5,000, 10,000, or 15,000 for **Rate** until the graph displays a smooth sine wave.
12. Click the **Analog Output** tab. On this page you can generate a DC voltage or sine wave on one of the analog output channels of the DAQ device. Complete the following steps to output a DC voltage on analog output channel 0.
 - a. Verify that analog out 0 is connected to analog in 2 on the DAQ Signal Accessory.
 - b. Select DC Voltage for the **Output Mode**.
 - c. Enter 5 V for the **Output Voltage/Amplitude** and click **Update**.
13. Click the **Analog Input** tab.
 - a. Change the channel to read from analog input channel 2 and set the **Acquisition Mode** to **On Demand**.
 - b. Click the **Start** button. You should see 5 V displayed on the graph.
 - c. Click the **Stop** button.
14. Click the **Digital I/O** tab to access the eight digital lines on the device. Use this page to set each line as an input or output line.



Note If you have an SCXI chassis configured, lines 4, 2, 1, and 0 are dimmed.

- a. Set lines 0 through 3 to be output lines. These lines correspond to the four LEDs on the DAQ Signal Accessory. Toggle these lines by checking and unchecking the **Logic Level**.

- b. Watch the LEDs turn on and off. The LEDs turn on when the Logic Level box is not selected because the LEDs in the DAQ Signal Accessory are based on reverse logic. In other words, the LED turns on when the digital line outputs a logic low.
15. Click the **Counter I/O** tab. Complete the following steps to verify counter/timer operation:
- a. Under **Counter Mode**, select the **Edge Counting** tab. The **Edge Source** defaults to `/DevX/20MHzTimeBase`, where *X* corresponds to the device number of your DAQ device. This counts the pulses of a 20 MHz onboard signal.
 - b. Click the **Start** button. The **Counter Value** should increment rapidly.
 - c. Click the **Stop** button to stop the counter test.
16. Click the **Stop** button.
17. Close the test panel and exit MAX.

End of Exercise 2-3

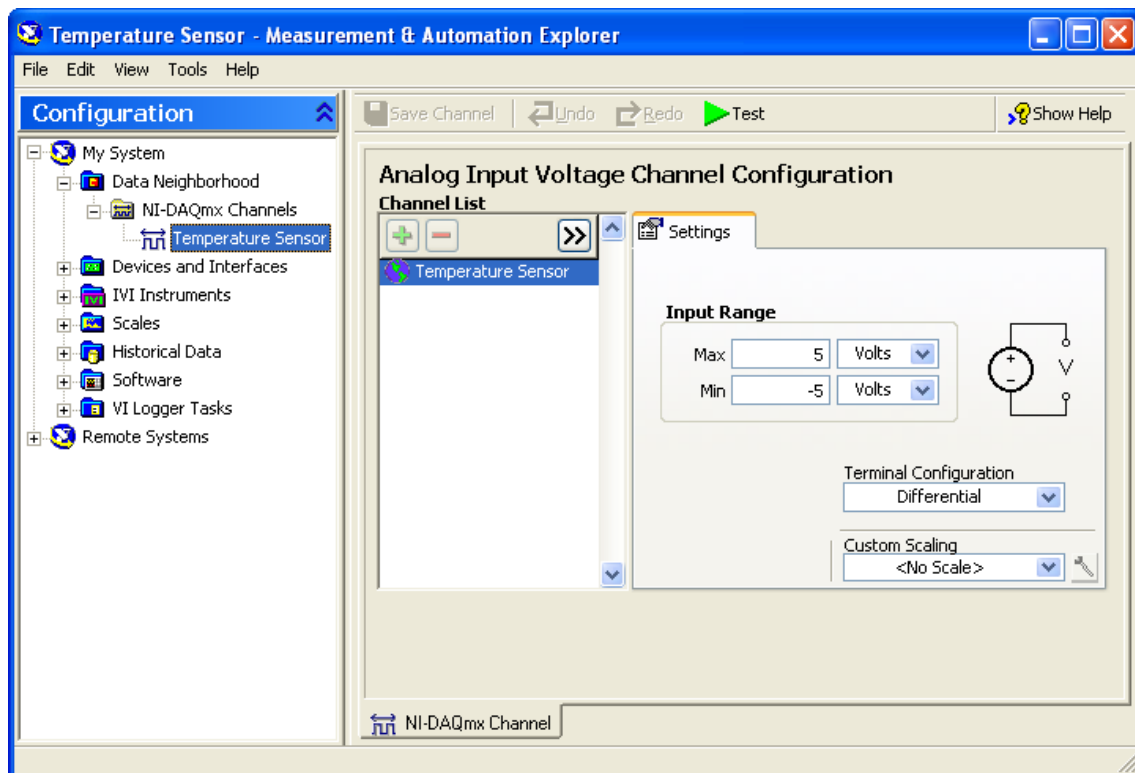
Exercise 2-4 DAQ Assistant

Objective: To create three NI-DAQmx channels using the DAQ Assistant and create an NI-DAQmx task from each channel. You also will create a custom scale to convert the units of the temperature sensor from voltage to degrees Celsius.

Part I—Creating Channels



1. Connect the sine wave from the DAQ Signal Accessory function generator to analog in 1 and the square waveform to analog in 2.
2. If you closed MAX in the previous exercise, restart it by double-clicking the icon on the desktop.
3. To configure channels with the DAQ Assistant, right-click **Data Neighborhood** and select **Create New**. Create a global channel for the temperature sensor on the DAQ Signal Accessory.
4. Select **NI-DAQmx Global Channel** and click the **Next** button.
5. In the dialog box that appears, select **Analog Input** as the measurement type.
6. Select **Voltage** as the sensor type. Although you take a temperature measurement for the temperature sensor on the DAQ Signal Accessory, do not select **Temperature** as the sensor type. Use the Temperature type when using a temperature-specific transducer, such as a thermocouple or RTD.
7. The next dialog box that appears prompts you to select the physical channel to use with your new virtual channel. Under **DevX**, where **X** corresponds to the device number of your DAQ device, select **ai0** and click **Next**.
8. Name the channel `Temperature Sensor`. Click the **Finish** button. The following channel configuration should appear in MAX.



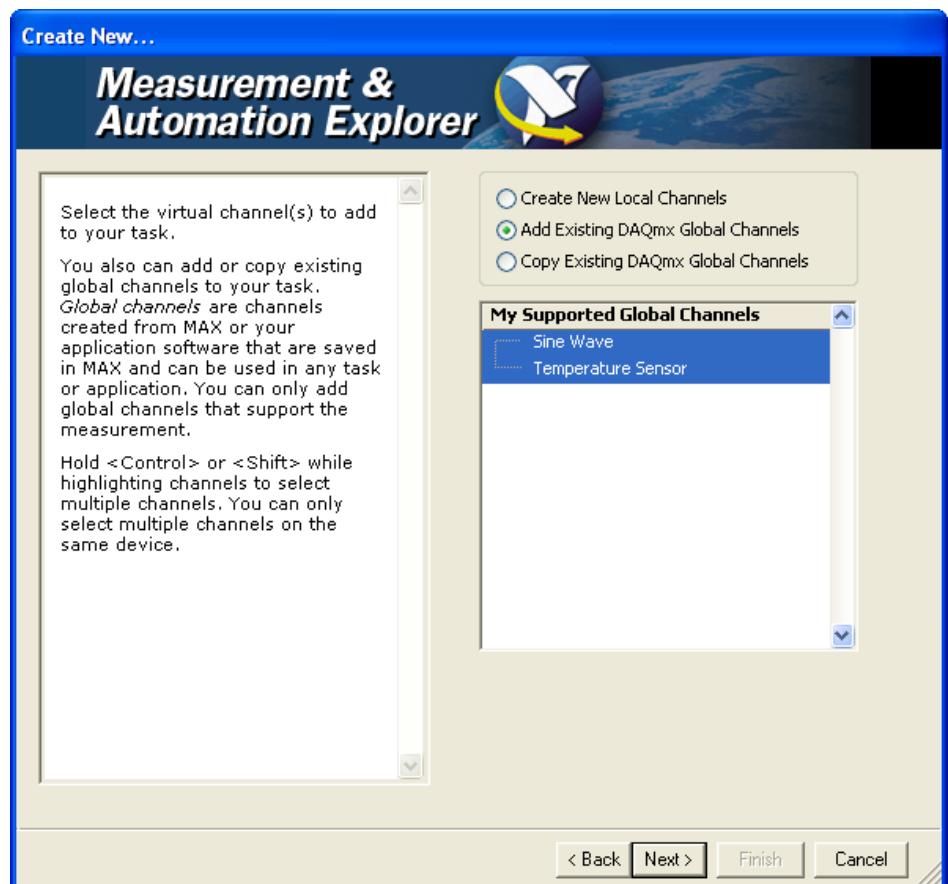
9. Click the **Test** button to launch a test panel for the channel. Click the **Start** button and verify that you are receiving data within 0.2 V to 0.3 V. Click the **OK** button to exit the test panel. After creating an NI-DAQmx channel in MAX, always test the channel before continuing to create other channels.
10. Repeat steps 3 to 8 to create an additional channel with the following settings:
 - **Second Channel—Sine Wave**
 - **Measurement Type:** Analog Input
 - **Sensor Type:** Voltage
 - **Physical Channel:** ai1
 - **Name:** Sine Wave

Part II—Creating a Task

In Part I of this exercise, you created two global channels. When two or more channels have similar timing and triggering needs, group the channels into a logical collection called an NI-DAQmx Task. Create a task and add the two previously created global channels and one local channel to the task.

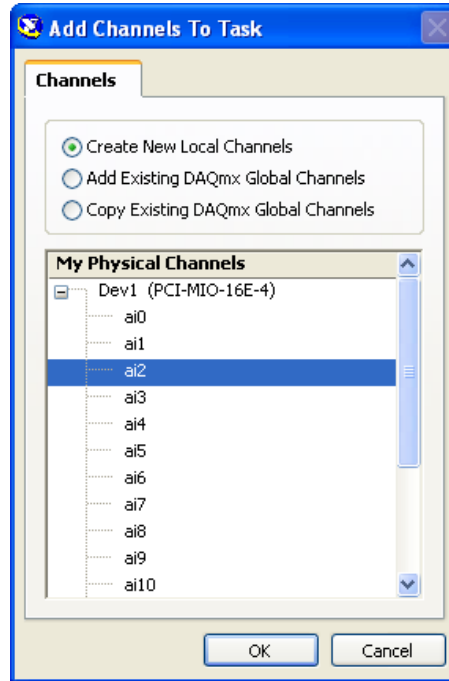
1. Right-click **Data Neighborhood** and select **Create New** from the shortcut menu.
2. Select **NI-DAQmx Task** and click the **Next** button.

3. Select **Analog Input** as the measurement type.
4. Select **Voltage** as the sensor type.
5. A dialog box prompts you to select the physical channels to include in the task. However, you want to use the global channels you created in Part I. To do so, select **Add Existing DAQmx Global Channels**.
6. Press and hold the <Shift> key and select both channels.



7. Click the **Next** button
8. Name the task `MyVoltageTask` and click the **Finish** button.
9. Review the **Analog Input Voltage Task Configuration** window. Right-click the **Temperature Sensor** channel and examine the shortcut menu. Notice how you can change the channel to a local channel.

10. Click the green + button and select **Create New Local Channels**. Select the physical channel ai2 for your DAQ device.

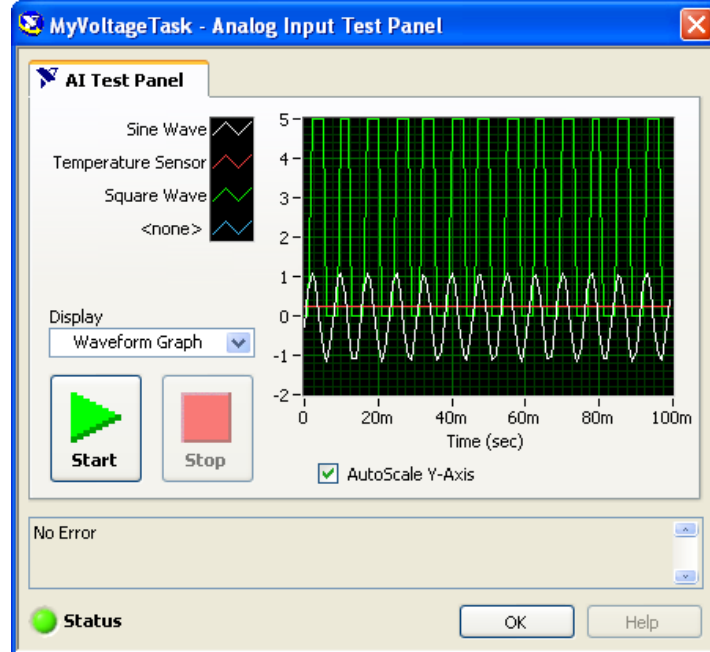


11. Click the **OK** button to create a channel named **Voltage**. To rename the channel, right-click the channel and select **Rename** from the shortcut menu. Rename the channel Square Wave.



12. Click the **OK** button to return to the DAQ Assistant. Notice that Square Wave channel does not have a globe icon next to it. This indicates that it is a local channel and can only be used by the MyVoltageTask task.

13. Click the **Test** button then click the **Start** button to acquire data.



14. Click the **OK** button to exit the test panel.

15. Click the **Save Task** button to save the task to MAX.

Part III—Custom Temperature Scale

The voltage values you are receiving for the temperature sensor on the DAQ Signal Accessory are in the range of 0.2 to 0.3 V. This voltage level multiplied by 100 corresponds to the temperature reading in degrees Celsius. Add a custom scale to the Temperature Sensor channel so a more readable value returns to the application.

1. Under **Data Neighborhood**, select **NI-DAQmx Global Channels** and select **Temperature Sensor**.
2. In the **Analog Input Voltage Channel** configuration, select **Custom Scaling** and select **Create New** from the shortcut menu.
3. Use the following settings for the custom scale:
 - **Scale Type:** Linear
 - **Name:** Temperature Scale
4. Click the **Finish** button. The DAQ Assistant appears. Configure the numerical scaling for your scale in the DAQ Assistant.
5. Since the voltage to degrees Celsius correspondence is one to 100, enter 100 for the **slope** value. Because there is no offset, leave the **y-intercept** value at the default, 0.

6. Set the **Scaled Units** to Deg C and set the **Pre-Scaled Units** to Volts.
7. Click the **OK** button.
8. For the Temperature Sensor channel, change the maximum and minimum input values to 40 and 0, respectively. For measuring the air temperature or the temperature of your finger, 0 to 40 degrees Celsius is an appropriate range.
9. Save the channel settings by clicking **Save Channel**.
10. Click the **Test** button and click the **Start** button to acquire data. Notice that the data is now in the range of twenty to thirty degrees Celsius.
11. Click the **OK** button to return to the DAQ Assistant.
12. Exit MAX.

End of Exercise 2-4

J. Overview of NI-DAQmx VIs

Now that you have an understanding of transducers, signals, DAQ device configuration, and MAX, you can begin learning to use LabVIEW to develop a data acquisition application. NI-DAQmx supports software packages other than LabVIEW, but this course describes only LabVIEW development of DAQ applications.

DAQmx Name Controls

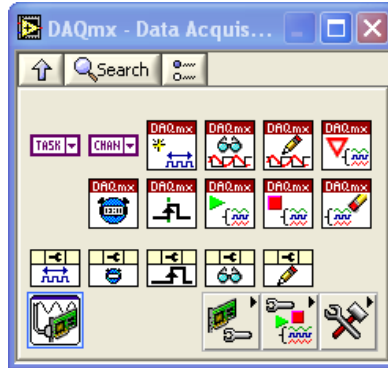
The DAQmx Name Controls are located on the **Controls»All Controls»I/O»DAQmx Name Controls** palette.



The **DAQmx Name Controls** palette includes controls for the task name, channel name, physical channel, terminal, scale name, device number, and switch. Because you typically configure tasks and channels in MAX prior to LabVIEW programming, you typically use only the Task Name and Channel Name controls. The values for the physical channel, terminal, scale name, and device number are configured in MAX when you create and configure a task or channel. The currently configured tasks and channels automatically populate the menu rings for the DAQmx Task Name and DAQmx Channel Name controls.

DAQmx - Data Acquisition VIs

Use the NI-DAQmx VIs located on the **Functions»All Functions»NI Measurements»DAQmx - Data Acquisition** palette with NI-DAQ and NI-SWITCH hardware devices to develop instrumentation, acquisition, and control applications. Refer to the *DAQ Quick Start Guide for NI-DAQ 7.0* or the *NI-DAQ 7.0 Readme* for a complete listing of devices that NI-DAQmx supports.



The **DAQmx - Data Acquisition** palette includes the following constants, VIs, Property Nodes, and subpalettes:

Constants

- DAQmx Task Name Constant—Lists all tasks you create and save using the DAQ Assistant. Right-click the constant and select **I/O Name Filtering** from the shortcut menu to limit the tasks the constant displays and to limit what you can enter in the constant.
- DAQmx Global Channel Constant—Lists all virtual channels you create and save by using the DAQ Assistant. Select **Browse** to select multiple channels. Right-click the constant and select **I/O Name Filtering** from the shortcut menu to limit the channels the constant displays and to limit what you can enter in the constant.

VIs

- DAQmx Create Virtual Channel VI—Creates a virtual channel or set of virtual channels and adds them to a task. The instances of this polymorphic VI correspond to the I/O type of the channel, such as analog input, digital output, or counter output; the measurement or generation to perform, such as temperature measurement, voltage generation, or event counting; and in some cases, the sensor to use, such as a thermocouple or RTD for temperature measurements.

This function sets the same settings that you configure in MAX when creating a virtual channel. Use this function if the operator of your program might periodically change the physical channel connection, but not other important settings, such as terminal configuration mode or the custom scale applied. Use the physical channel pull-down menu to specify the device number of the DAQ device and the actual physical channel your signal is connected to.

- DAQmx Read VI—Reads samples from the task or channels you specify. The instances of this polymorphic VI specify what format of samples to return, whether to read a single sample or multiple samples at once, and whether to read from one or multiple channels.

- DAQmx Write VI—Writes samples to task or channels you specify. The instances of this polymorphic VI specify the format of the samples to write, whether to write one or multiple samples, and whether to write to one or multiple channels.
- DAQmx Wait Until Done VI—Waits for the measurement or generation to complete. Use this VI to ensure that the specified operation is complete before you stop the task.
- DAQmx Timing VI—Configures number of samples to acquire or generate and creates a buffer when needed. The instances of this polymorphic VI correspond to the type of timing to use on the task.
- DAQmx Trigger VI—Configures triggering for the task. The instances of this polymorphic VI correspond to the trigger and trigger type to configure.
- DAQmx Start Task VI—Transitions the task to the running state to begin the measurement or generation. Using this VI is required for some applications and is optional for others.
- DAQmx Stop Task VI—Stops the task and returns it to the state the task was in before you used the DAQmx Start Task VI or used the DAQmx Write VI with the **autostart** input set to TRUE.
- DAQmx Clear Task VI—Clears the task. Before clearing, this VI stops the task, if necessary, and releases any resources reserved by the task. You cannot use a task after you clear it unless you recreate the task.

Refer to the *NI-DAQmx Task State Model* section of this lesson for more information about the DAQmx Task State Model.

Property Nodes

- DAQmx Channel Property Node—A Property Node with the DAQmx Channel class preselected. Right-click the Property Node and choose **Select Filter** from the shortcut menu to make the Property Node show only the properties supported by a particular device installed in the system or supported by all the devices installed in the system.
- DAQmx Timing Property Node—A Property Node with the DAQmx Timing class preselected. Right-click the Property Node and choose **Select Filter** from the shortcut menu to make the Property Node show only the properties supported by a particular device installed in the system or supported by all the devices installed in the system.
- DAQmx Trigger Property Node—A Property Node with the DAQmx Trigger class preselected. Right-click the Property Node and choose **Select Filter** from the shortcut menu to make the Property Node show only the properties supported by a particular device installed in the system or supported by all the devices installed in the system.

- DAQmx Read Property Node—A Property Node with the DAQmx Read class preselected. Right-click the Property Node and choose **Select Filter** from the shortcut menu to make the Property Node show only the properties supported by a particular device installed in the system or supported by all the devices installed in the system.
- DAQmx Write Property Node—A Property Node with the DAQmx Write class preselected. Right-click the Property Node and choose **Select Filter** from the shortcut menu to make the Property Node show only the properties supported by a particular device installed in the system or supported by all the devices installed in the system.

DAQ Assistant

- DAQ Assistant Express VI—Creates, edits and runs tasks using NI-DAQmx. Refer to the *DAQ Quick Start Guide for NI-DAQ 7.0* for a complete listing of devices NI-DAQmx supports.

Subpalettes

- Use the DAQmx Device Configuration VIs and functions located on the **DAQmx Device Configuration** palette for hardware specific configuration and control.
- Use the DAQmx Advanced Task Options VIs and functions located on the **DAQmx Advanced Task Options** palette for advanced configuration and control of tasks.
- Use the DAQmx Advanced VIs and Functions located on the **DAQmx Advanced** palette to access advanced and miscellaneous features of NI-DAQmx.

K. NI-DAQmx Task State Model

NI-DAQmx uses a task state model to control the resource allocation and execution flow of tasks. You call the DAQmx Start VI, DAQmx Stop VI, and DAQmx Control Task VIs to transition the task from one state to another. The task state model is very flexible. You can choose to interact with as little or as much of the task state model as your application requires. You can explicitly transition between each task using the DAQmx Control Task VI, or you can allow NI-DAQmx to perform the state transitions implicitly. The task state model consists of the following five states:

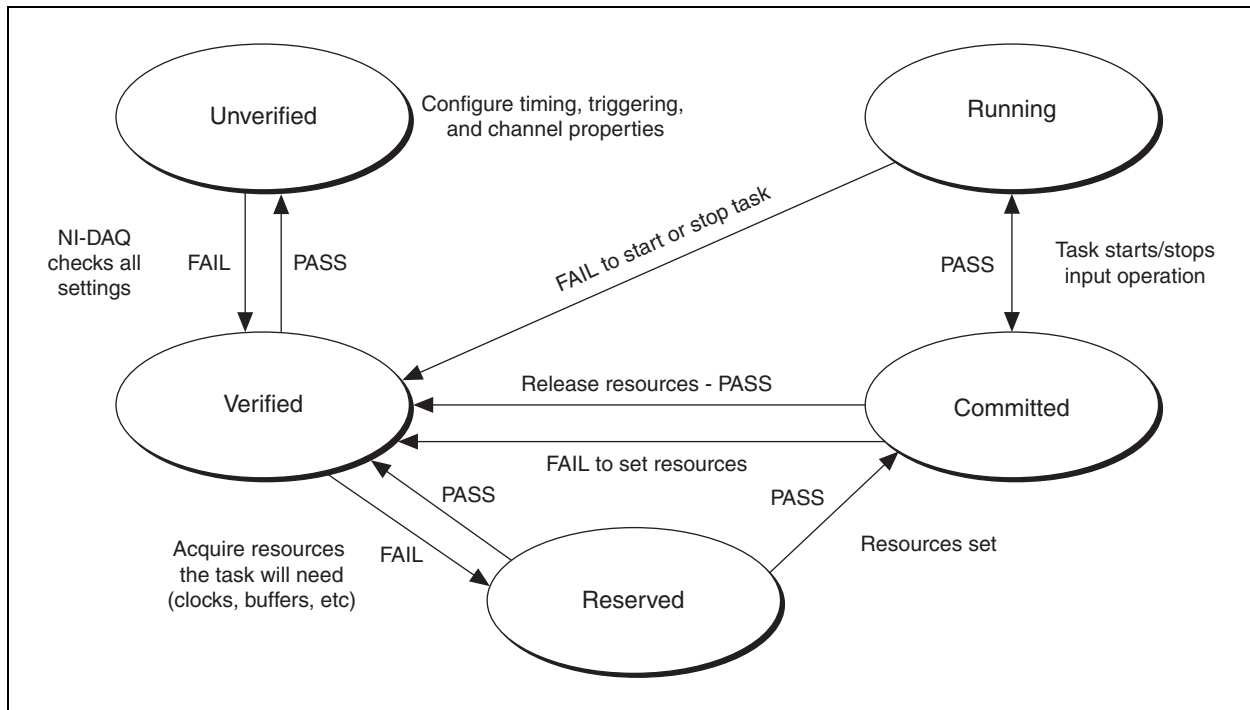
- **Unverified**—When a task is created or loaded, either explicitly or implicitly, it is in the Unverified state. In this state, you configure the timing, triggering, and channel properties of the task.
- **Verified**—NI-DAQ checks the timing, triggering, and channel properties for correctness when the task transitions from the Unverified to the Verified state. You can explicitly perform this transition by calling the DAQmx Control Task VI with the **action** input set to verify. While NI-DAQmx detects and verifies some invalid values for properties/attributes immediately when you set the property/attribute, NI-DAQmx cannot verify other values immediately because they depend on other properties/attributes and the devices being used. NI-DAQmx checks the values of these properties/attributes during the verify transition and reports any invalid values at that time. If NI-DAQmx finds no invalid values, the task successfully verifies and transitions to the Verified state. Otherwise, it remains in the Unverified state.
- **Reserved**—The resources a task uses to perform the specified operation are acquired exclusively when the task transitions from the Verified state to the Reserved state. These resources can be clocks or channels on a device, trigger lines on a PXI chassis, or buffer memory in the computer. Reserving these resources prevents other tasks from using these resources, which interferes with this task performing the specified operation. You can explicitly perform this transition by calling the DAQmx Control Task VI with the **action** input set to reserve. This transition fails if some task resources are currently reserved by another task. If the task can gain access to all the resources it uses, the task is successfully reserved and transitions to the Reserved state. Otherwise, it remains in the Verified state.
- **Committed**—NI-DAQmx programs some of the settings for the resources when the task is committed. These settings might be the rate of the clock or the input limits of a channel on a device, the direction of a trigger line on a PXI chassis, or the size of the buffer memory in the computer. Other settings, such as the sample counter, cannot be programmed when the task is committed because they need to be

programmed every time the task starts. When a task is committed, it transitions from the Reserved state to the Committed state. You can explicitly perform this transition by calling the DAQmx Control Task VI with the **action** input set to commit. In general, the commit transition should not fail. If it does, it is an exceptional condition and the task remains in the Reserved state. If the settings for the resources used by the task are programmed, the task is successfully committed and transitions to the Committed state.

- **Running**—When the task begins to perform the specified operation, the task transitions from the Committed state to the Running state. You can explicitly perform this transition by calling the DAQmx Start VI. Notice that starting a task does not necessarily start acquiring samples or generating a waveform. You might have specified the timing and triggering properties such that a sample is not acquired until a trigger is detected. In general, the start transition does not fail. If it does, it is an exceptional condition and the task remains in the Reserved state. This returns the task to the Verified state. If the task begins to perform the specified operation, the task is successfully started and transitions to the Running state.

If you explicitly invoke a state transition that has already occurred, it is not repeated and an error is not returned. For example, if the task has already reserved its resources, and therefore is in the Reserved state, calling the DAQmx Control Task VI with the **action** input set to reserve does not reserve the resources again.

The following illustration summarizes the task state model.



Explicit Versus Implicit State Transitions

When should you perform explicit state transitions, and when should you rely on the task to perform implicit state transitions? The answer depends on your application. The following list identifies instances in which you should use explicit state transitions:

- **Verify**—If in your application users interactively configure a task by setting various channel, timing, and triggering properties, explicitly verify the task occasionally to inform the users if they have set a property to an invalid value.
- **Reserve**—If the following is true, explicitly reserve a task: your application contains many different tasks that use the same set of resources, one of these tasks repeatedly performs its operation, and you want to ensure that none of the other tasks acquires these resources after the task begins its sequence of operations.

Reserving the task exclusively acquires the resources that the task uses, ensuring that other tasks cannot acquire these resources. For example, if your application contains two tasks that each perform a sequence of measurements and you want to ensure that each sequence is completed before the other sequence begins, you can explicitly reserve each task before it begins its sequence of measurements.

- **Commit**—If your application performs multiple measurements or generations by repeatedly starting and stopping a task, explicitly commit a task. Committing the task exclusively acquires the resources that the task uses and programs some of the settings for these resources. By explicitly committing the task, these operations are performed once, not each time the task starts, which can considerably decrease the time needed to start the task. For example, if your application repeatedly performs finite, hardware-timed measurements, the time required to start the task can dramatically decrease if you explicitly commit the task before repeatedly performing these measurements. Explicitly committing a task also is required if you need to perform additional read operations of the samples acquired by the task after stopping the task.
- **Start**—If your application repeatedly performs read or write operations, explicitly start a task. Starting the task reserves the resources that the task uses, programs some of the settings for these resources, and begins to perform the specified operation. By explicitly starting the task, these operations are performed once, not each the time read or write operation is performed. This process can considerably decrease the time required to perform each read or write operation. For example, if your application repeatedly performs single-sample, software-timed read operations, the time required for each read operation can dramatically decrease if you explicitly start the task before repeatedly performing these read operations.

Implicit State Transitions

Although you can explicitly transition a task through each of its states, you rarely need this level of detailed control. Two scenarios exist in which a task implicitly transitions from one state to another:

- **Task Moves Through Multiple States at the Same Time**—Some state transitions require the task to move through one or more states to reach the specified state. For example, if the task is in the Unverified state and you call the DAQmx Control Task VI, setting the action input to reserve, the task is verified and reserved. The task transitions from the Unverified state to the Verified state and to the Reserved state. In most applications, it is not helpful to explicitly transition the task to each state. Instead, invoke only those transitions that are necessary, and the task implicitly handles the rest.

- **Operations that Require State Transitions**—You implicitly transition the task to a new state when you perform an operation that requires that the task be in a specific state and it is not. If this occurs, the task implicitly transitions to the required state. Some operations that require state transitions include the following:
 - Querying the value of a property/attribute implicitly verifies the task. This verification is required to return accurate coerced values of properties. Because the coerced value of a property/attribute often depends on the values of other properties/attributes, the task as a whole must be verified to calculate the value. Because the task might be implicitly verified when you query the value of a property/attribute, NI-DAQ might return an error specifying that the value of the property/attribute is invalid.
 - Calling the DAQmx Read VI implicitly commits the task if the task is not already committed. If the value of the Auto Start property is True and the task has not started, the task also implicitly starts.
 - Calling the DAQmx Write VI implicitly commits the task. If the value of the Auto Start property is True, the task also implicitly starts.

For example, if the task is in the Reserved state, the value of the Auto Start property is True, and you call the DAQmx Read VI, the task implicitly commits and starts. The task transitions from the Reserved state to the Committed state and to the Running state before performing the read operation.

In some applications, it is not necessary to explicitly transition the task to any state. Instead, invoke the operation you want and the task implicitly handles everything else.

Summary

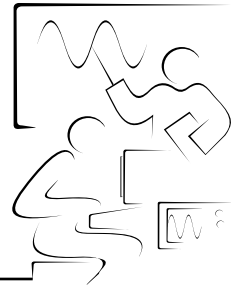
- Typical DAQ hardware consists of a terminal block, a cable, and a DAQ device.
- Typical DAQ devices have connectors, ADC, DAC, digital I/O lines, and counters.
- Code width is the smallest voltage change a DAQ device can detect based on resolution, range, and gain.
- DAQ devices can be grounded in the following modes:
 - Differential
 - RSE
 - NRSE
- LabVIEW communicates with DAQ devices using NI-DAQ software.
- You can use MAX to access the DAQ Assistant, view test panels, and obtain software updates to help configure and test the system.
- You can use the DAQ Assistant from MAX or LabVIEW to create and edit virtual channels, tasks, and custom scales.
- The **DAQmx - Data Acquisition** palette contains all the VIs necessary to perform customized data acquisition and generation.
- The Task State Model allows for a detailed level of control over a DAQmx task. For most applications, implicit transitions between states is sufficient, although you can explicitly perform state transitions using the DAQmx Control Task VI.

Notes

Notes

Lesson 3

Triggering



This lesson describes the theory and concepts of analog and digital triggering.

You Will Learn:

- A. Analog and digital triggering
- B. Types of triggers
- C. How to use the DAQ Assistant to test and configure triggers

A. 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 example, 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.

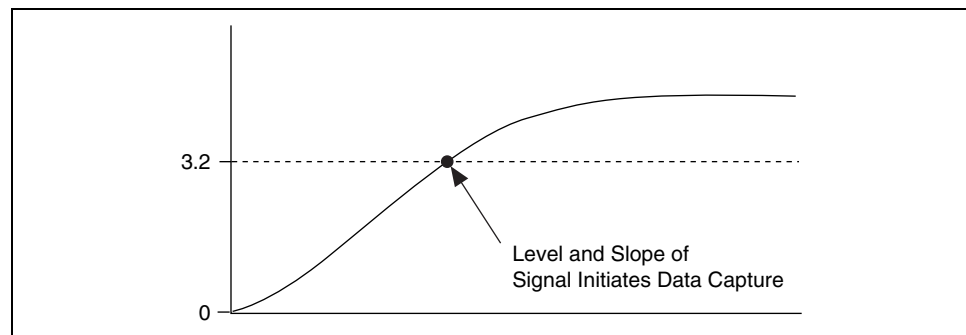
When you configure a trigger, you must make two main decisions—what action you want the trigger to cause and how to produce the trigger.

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. For example, consider an application that monitors a temperature system. If you want to begin data acquisition only after the temperature rises to 50 °C, configure an analog trigger to occur when the temperature signal has a rising slope and voltage level corresponding to 50 °C. The following illustration shows triggering on a rising or falling slope at a level of 2.7 V.

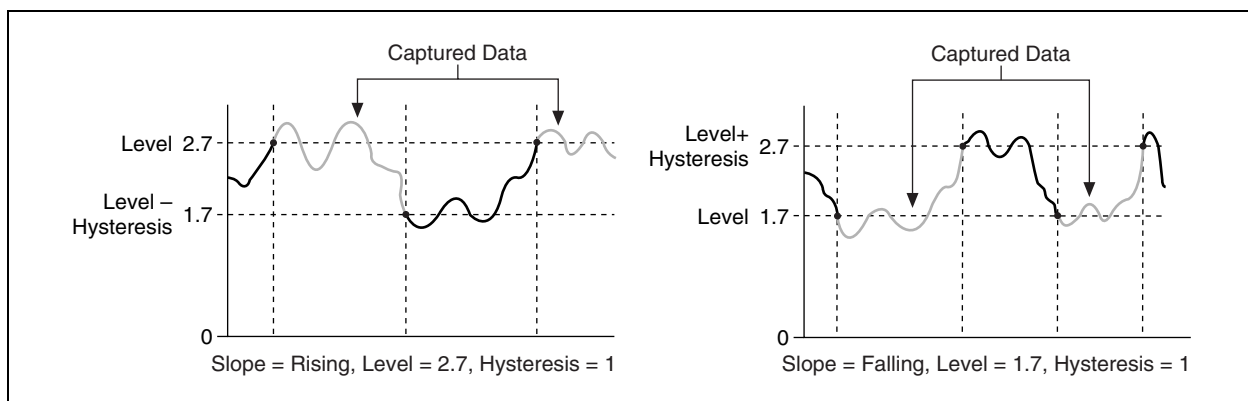


Hysteresis

Hysteresis adds a window above or below the trigger level and often is used to reduce false triggering due to noise or jitter in the signal. When using hysteresis with a rising slope, the trigger asserts when the signal starts below **level** (or **threshold level**) and then crosses above **level**. The trigger deasserts when the signal crosses below **level** minus **hysteresis**.

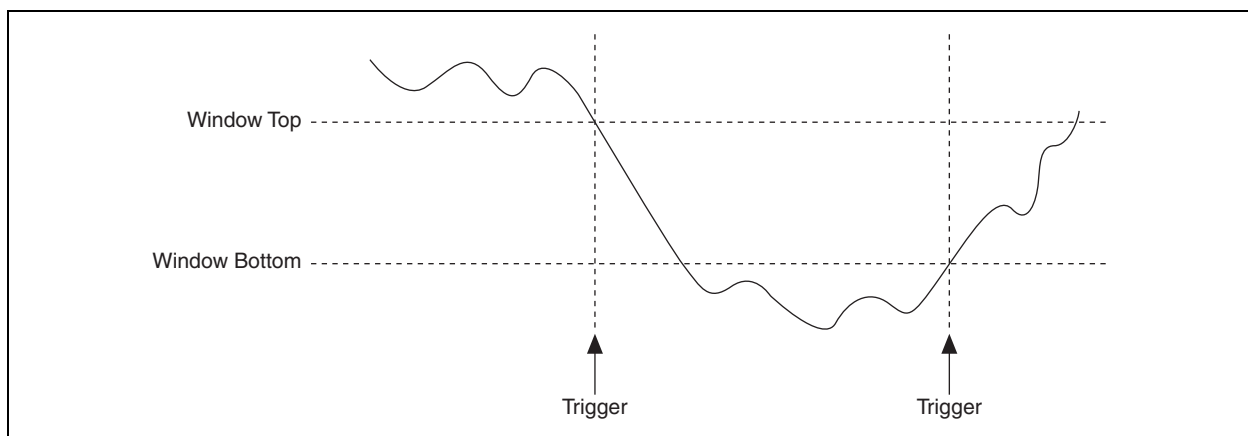
When using hysteresis with a falling slope, the trigger asserts when the signal starts above **level** (or **threshold level**) and then crosses below **level**. The trigger deasserts when the signal crosses above **level** plus **hysteresis**.

The following illustration demonstrates the data captured when using hysteresis with a rising and falling edge slope at a level of 2.7 V.

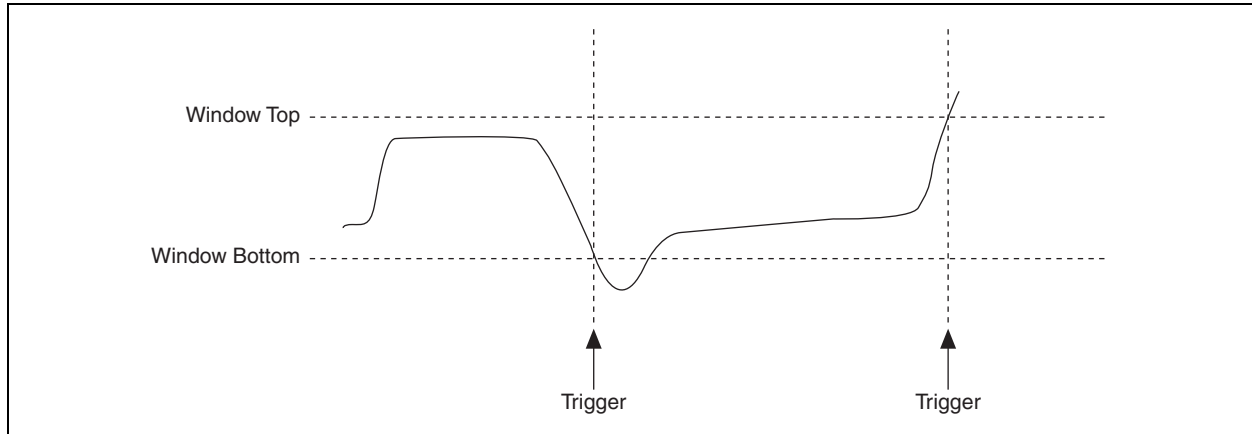


Analog Window Triggering

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 the following illustration, the trigger acquires data when the signal enters the window.

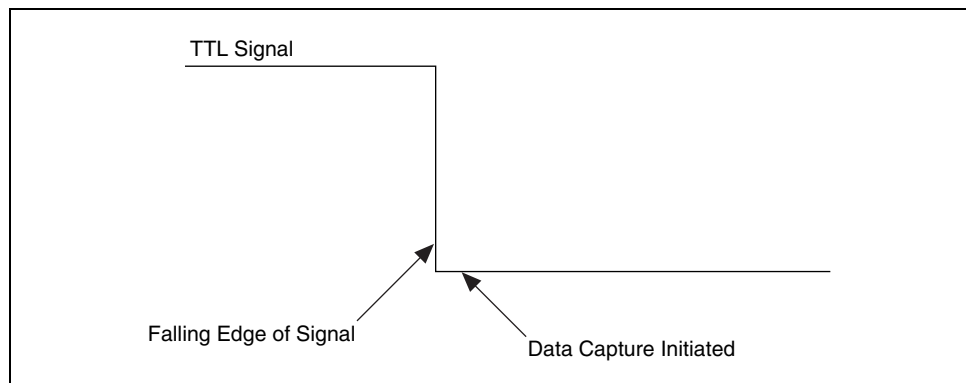


In the following illustration, the trigger acquires data when the signal leaves the window.



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 or falling edge of a digital signal as shown in the following illustration. You usually connect digital trigger signals to PFI pins in a National Instruments measurement device.



Actions Caused by Triggering

There are four actions that a trigger can cause. Triggers are named after the actions they cause:

- **Advance Trigger**—Causes a switch device to execute the next entry in its instruction (scan) list.
- **Pause Trigger**—Pauses an acquisition. Deasserting this trigger resumes an acquisition.
- **Reference Trigger**—Establishes the reference point in a set of input samples. Data acquired up to the reference point is pretrigger data. Data acquired after this reference point is posttrigger data.

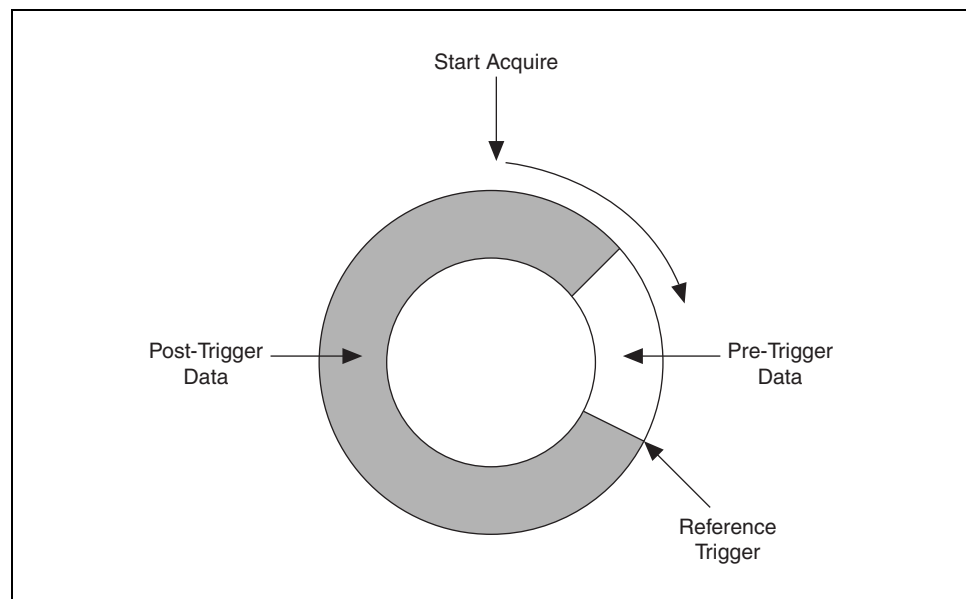


Figure 3-1. Reference Trigger

- **Start Trigger**—Begins an acquisition or generation.

This course describes only start, reference, and pause triggers.



Note Not all E Series devices support analog triggering. Refer to your device documentation to determine if your device supports analog triggering.

Exercise 3-1 Triggering using the DAQ Assistant in LabVIEW

Objective: To use the DAQ Assistant in LabVIEW to explore and configure different types of analog and digital triggers.

1. Connect the sine wave from the function generator to analog in 1 and the square wave from the function generator to analog in 2.
2. Launch LabVIEW and open a blank VI.

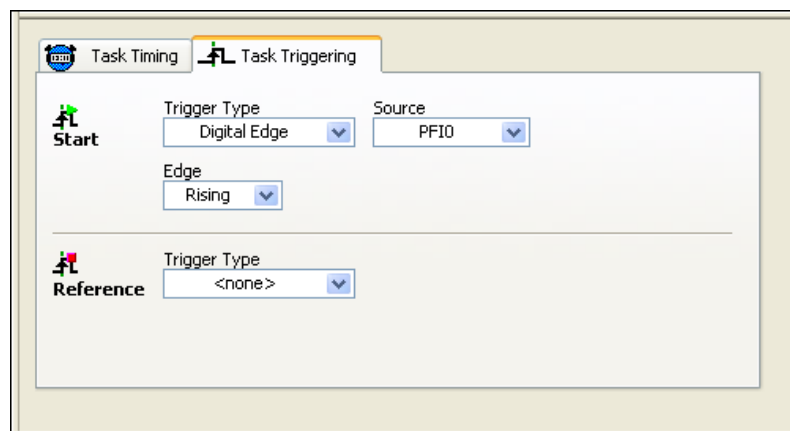
Front Panel

3. Place a DAQmx Task Name control, located on the **Controls»All Controls»I/O»DAQmx Name Controls** palette, on the front panel. Right-click the control and select **New Task (DAQ Assistant)** from the shortcut menu.
4. Use the DAQ Assistant to create a new task with the following settings:
 - **Measurement Type:** Analog Input
 - **Sensor Type:** Voltage
 - **Channels:** Select **Create New Local Channels** and highlight channels ai0 and ai1 under the appropriate DAQ device by holding down the <Shift> key.
 - **Name:** Trigger Task
5. Click the **Finish** button.

Start Triggers

The following steps demonstrate digital edge and analog edge start triggers.

1. In the DAQ Assistant, click the **Task Triggering** tab. In the **Start** section, select **Digital Edge** from the **Trigger Type** pull-down menu.
2. Select **PFIO** from the **Source** pull-down menu and select **Rising** from the **Edge** pull-down menu. The digital trigger button on the DAQ Signal Accessory corresponds to the PFIO/TRIG1 signal line.

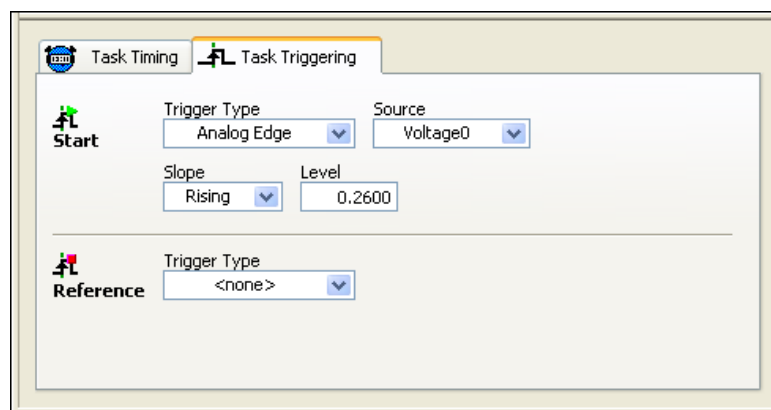


3. Click the **Test** button.
 4. The test panel is empty because the task is waiting on the rising edge of the PFIO line before acquiring any data. Press the digital trigger button on the DAQ Signal Accessory to enable the trigger.
 5. Change the Y-axis scale to a minimum of 0.2 and a maximum of 0.3. Notice the approximate value of the channel Voltage0 and record it below. This is the raw voltage off the temperature sensor. Click the **OK** button to exit the test panel.
-
6. Click the >> button above the channel list.
 7. Change the input limits of Voltage0 to 0 and -1.
 8. Because you chose analog input channels 0 and 1 for the task, the channels are listed in increasing order. The first channel in the scan order can trigger the remaining channels. Setting the ai0 channel as the first channel in the scan order indicates to NI-DAQmx that you want to trigger **Trigger Task** off of the temperature sensor channel of the DAQ Signal Accessory.



Note Steps 9 to 13 require an AI triggering DAQ device. If your computer does not have an AI triggering DAQ device, skip to the *Reference Triggers* section.

9. In the DAQ Assistant, click the **Task Triggering** tab. In the **Start** section, select **Analog Edge** from the **Trigger Type** pull-down menu.
10. Select **Voltage0** from the **Source** pull-down menu and select **Rising** from the **Slope** pull-down menu. Set **Level** slightly higher than the ambient temperature reading you noted in step 5.



11. Click the **Test** button.
12. Place your index finger over the temperature sensor to increase the temperature. When the temperature reaches the value of **Level**, data acquisition begins.



Note If triggering is not working, try changing the input limits of Voltage0 to a narrower input voltage range to decrease the code width and improve resolution of the triggering signal.

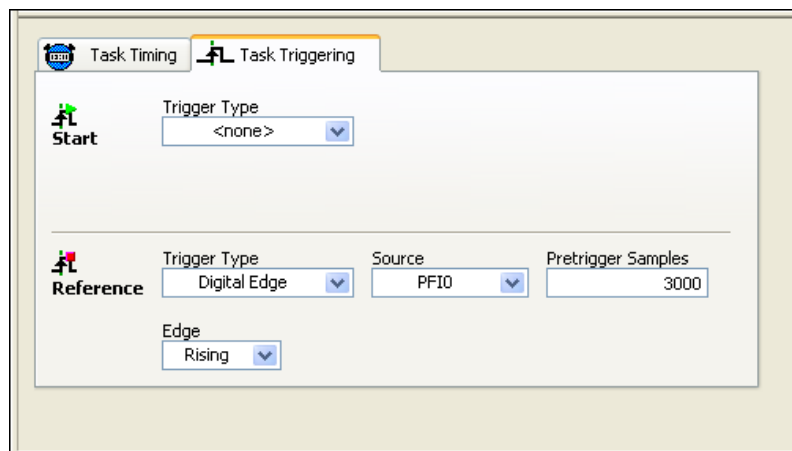
13. After acquiring data, click the **Stop** button and click the **OK** button to exit the test panel.

Reference Triggers

The following steps demonstrate using reference triggers.

1. Wire output A of the quadrature encoder to analog in 1 on the DAQ Signal Accessory.
2. Click the red – sign and remove analog input channel 0 from the task.
3. Click the **Task Timing** tab. Configure the following settings:
 - Acquire N Samples
 - **Samples to Read:** 5000
 - **Rate (Hz):** 1000
 - **Advanced Clock Settings - Clock Type:** Internal
4. Click the **Task Triggering** tab. In the **Start** section, select **none** from the **Trigger Type** pull-down menu.
5. In the **Reference** section, select **Digital Edge** from the **Trigger Type** pull-down menu and select **PFIO** from the **Source** pull-down menu.

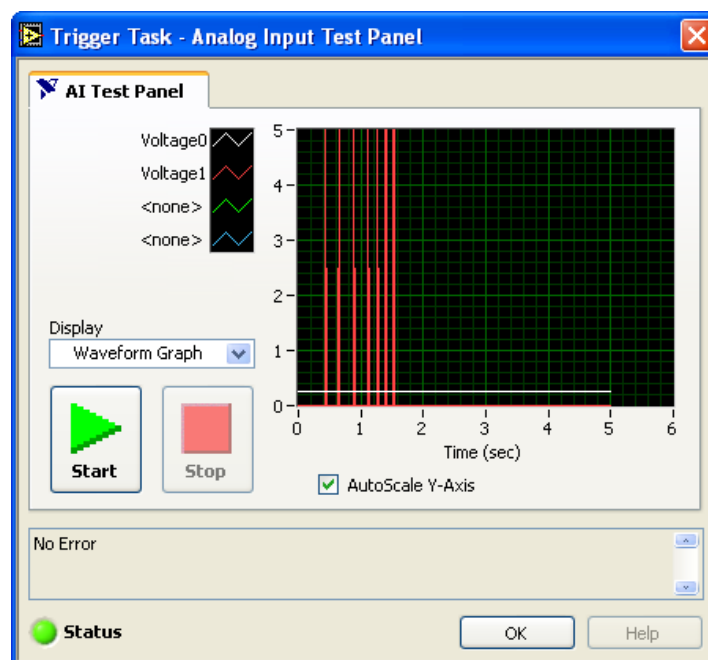
Pretrigger Samples is the minimum number of pretrigger samples to acquire before recognizing a reference trigger. The minimum number of pretrigger samples allowed by NI-DAQmx is 2. The number of posttrigger samples is equal to the **Samples to Read** (specified in the **Task Timing** tab) minus the number of **Pretrigger Samples**.



Set **Pretrigger Samples** to 3000. Since the total number of samples to acquire is 5,000 and the sampling rate is 1,000 Hz, the first three seconds will acquire the pretrigger samples.

6. Click the **Test** button.
7. Rotate the quadrature encoder for approximately three seconds then press the digital trigger button on the DAQ Signal Accessory. The number of pulse spikes displayed on the test panel corresponds to the number of clicks the quadrature encoder rotated.

In the following example, the quadrature encoder rotated three clicks before the digital trigger button was pressed. Each click corresponds to a 5 V spike in the graph.



8. Click the **OK** button to exit the test panel.
9. Click the **OK** button to exit the DAQ Assistant.

NI-DAQmx Code Generation

The following steps demonstrate using the code generation feature of NI-DAQmx to automatically generate LabVIEW code based on settings for timing or triggering in a task or channel you configured in the DAQ Assistant.

1. Right-click the **DAQmx Task Name** control on the front panel and select **Generate Code»Configuration and Example** from the shortcut menu. This option generates LabVIEW code to configure the task based on the settings you selected in the DAQ Assistant. This option also generates code for an example measurement. In this case, the example measurement is an analog input since that is the type of measurement you selected for the task.
2. View the front panel and watch the code automatically appear. A waveform graph appears on the front panel because you configured the task to return N samples.
3. Run the VI. Rotate the quadrature encoder for approximately three seconds then press the digital trigger button on the DAQ Signal Accessory.
4. Close the VI and exit LabVIEW. Do not save the VI.

End of Exercise 3-1

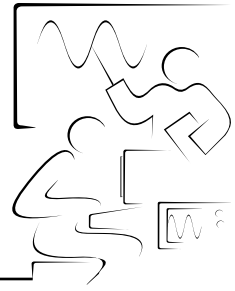
Summary

- Triggers can cause start, reference, pause, or advance actions.
- You can trigger off a digital edge.
- You can trigger off an analog edge or window.
- Many DAQ devices support analog triggering.
- Analog triggering is not a computationally intensive operation.
- Use the DAQ Assistant to test and configure triggering for NI-DAQmx tasks and channels. You also can use the DAQ Assistant to generate code in LabVIEW.

Notes

Lesson 4

Analog Input



This lesson describes decisions you must make to sample an analog input signal and the LabVIEW features that you use specifically with the NI-DAQmx VIs.

You Will Learn:

- A. Analog input considerations
- B. Anti-aliasing filters
- C. How to use the DAQmx Read VI
- D. Single-point acquisition
- E. Buffered acquisition
- F. Triggered acquisition

A. Analog Input

Because of the many advantages of digital signal processing, analog signals also are converted to digital form before they are processed with a computer. A digital signal is one that can assume only a finite set of values in both the dependent and independent variables. The independent variable is usually time or space, and the dependent variable is usually amplitude.

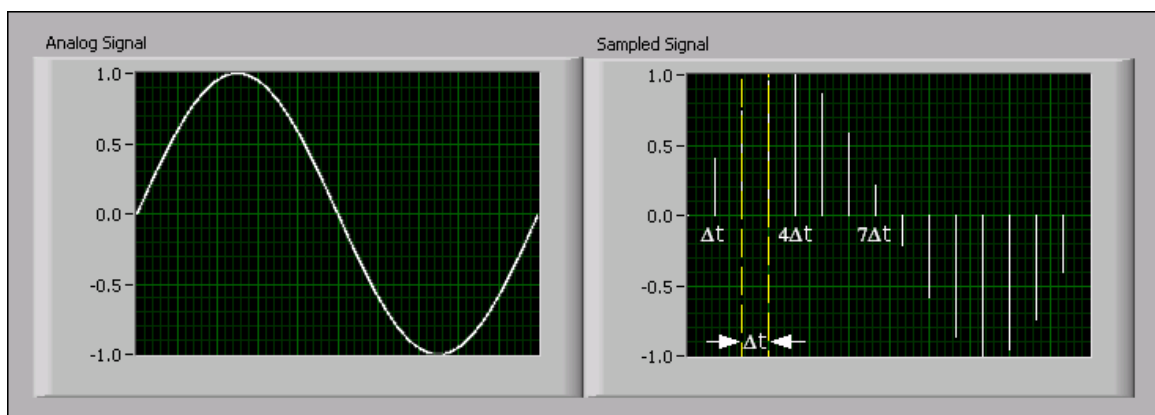
Digital signals are everywhere in the world around us. Telephone companies use digital signals to represent the human voice. Radio, TV, and hi-fi sound systems are all gradually converting to the digital domain because of its superior fidelity, noise reduction, and signal processing flexibility. Data is transmitted from satellites to earth ground stations in digital form. NASA's pictures of distant planets and outer space often are processed digitally to remove noise and to extract useful information. Economic data, census results, and stock market prices are all available in digital form.

Sampling Signals

To acquire an analog signal, you must first convert an analog signal into its digital representation. In practice, this is implemented by using an analog-to-digital (A/D) converter. Consider an analog signal $x(t)$ that is sampled every Δt seconds. The time interval Δt is known as the sampling interval or sampling period. Its reciprocal, $1/\Delta t$, is known as the sampling frequency, with units of samples/second. Each of the discrete values of $x(t)$ at $t = 0, \Delta t, 2\Delta t, 3\Delta t$, etc., is known as a sample. Thus, $x(0), x(\Delta t), x(2\Delta t), \dots$, are all samples. The signal $x(t)$ can be represented by the discrete set of samples as shown in the following equation.

$$\{x(0), x(\Delta t), x(2\Delta t), x(3\Delta t), \dots, x(k\Delta t), \dots\}$$

The following figure shows an analog signal and its corresponding sampled version. The sampling interval is Δt . The samples are defined at discrete points in time.



In this course, the following notation represents the individual samples:

$$x[i] = x(i\Delta t), \quad \text{for } i = 0, 1, 2, \dots$$

If N samples are obtained from the signal $x(t)$, $x(t)$ can be represented by the sequence:

$$X = \{x[0], x[1], x[2], x[3], \dots, x[N-1]\}$$

This is known as the digital representation or the sampled version of $x(t)$. Notice that the sequence $X = \{x[i]\}$ is indexed on the integer variable i , and does not contain any information about the sampling rate. By knowing just the values of the samples contained in X , you will not know what the sample rate is.

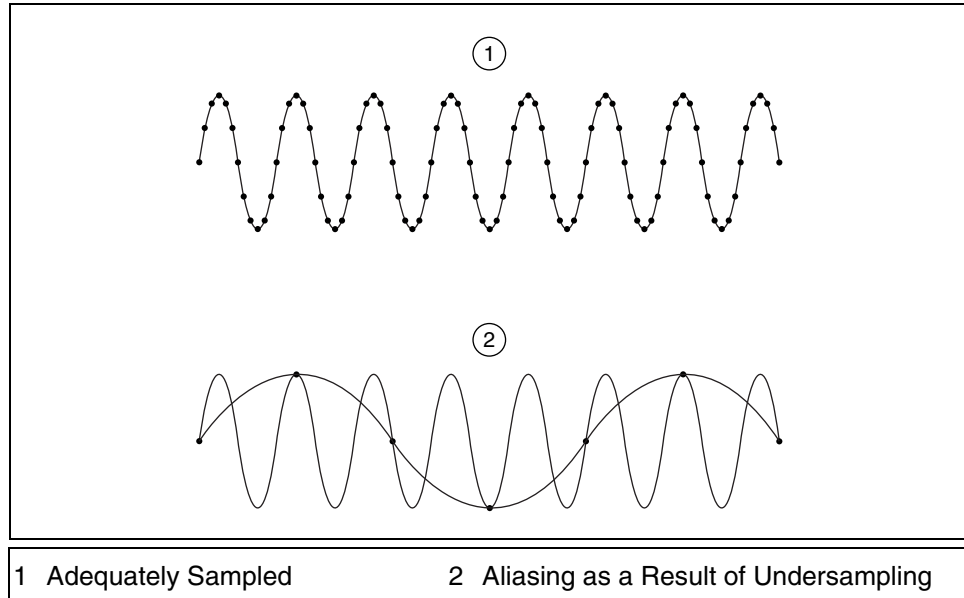
Sampling Rate

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 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 sampling rate can. Generating a 1 Hz signal using 1,000 points per cycle at 1,000 S/s produces a much finer representation than using 10 points per cycle at a sample rate of 10 S/s.

Aliasing

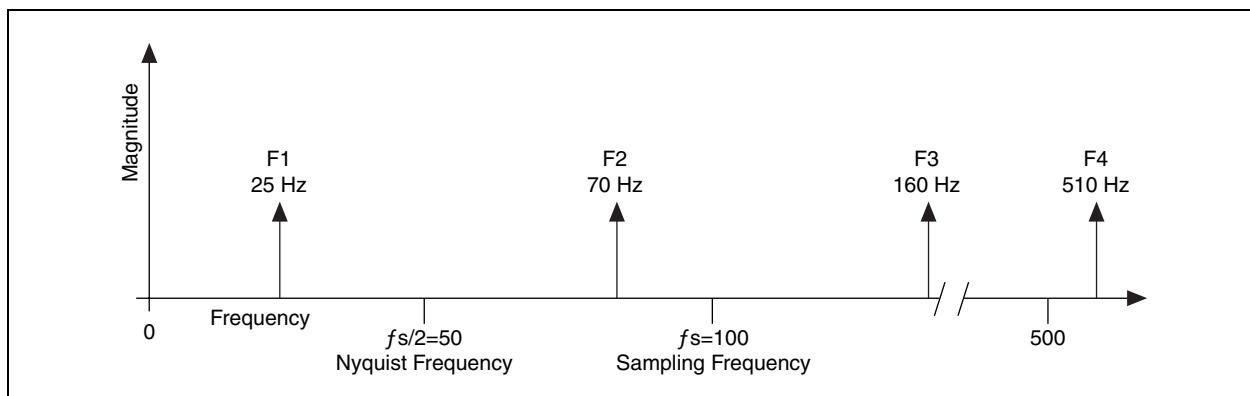
Sampling too slowly results in aliasing, which is a misrepresentation of the analog signal. Undersampling causes the signal to appear as if it has a different frequency than it actually does. To avoid aliasing, sample several times faster than the frequency of the signal.

The following illustration shows an adequately sampled signal and the aliasing effects of undersampling.

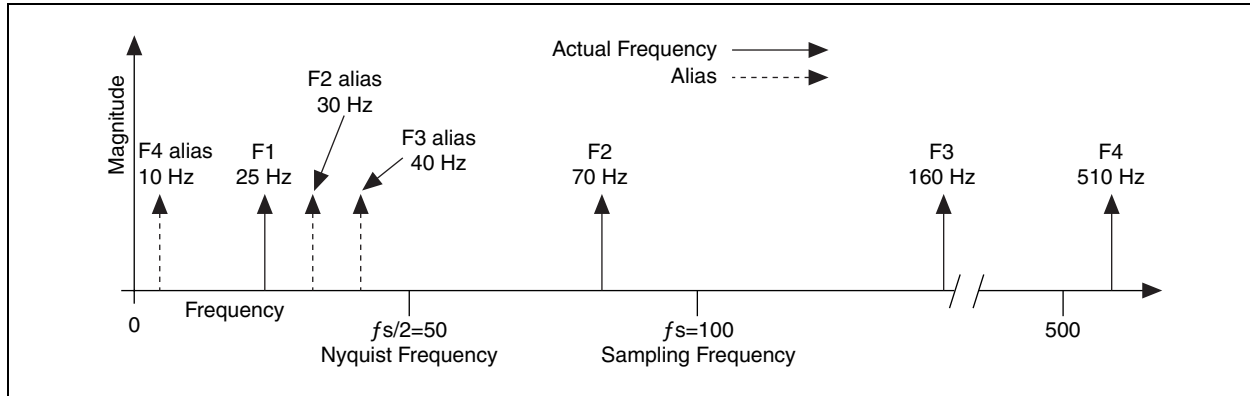


For frequency measurements, according to the Nyquist theorem, you must sample at a rate greater than twice the maximum frequency component in the signal you are acquiring to accurately represent the signal. The Nyquist frequency is the maximum frequency you can represent accurately without aliasing for 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, f_s , 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 the following illustration.



Frequencies below the Nyquist frequency ($f_s/2 = 50$ Hz) are sampled correctly, as shown in the following illustration. 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.



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,

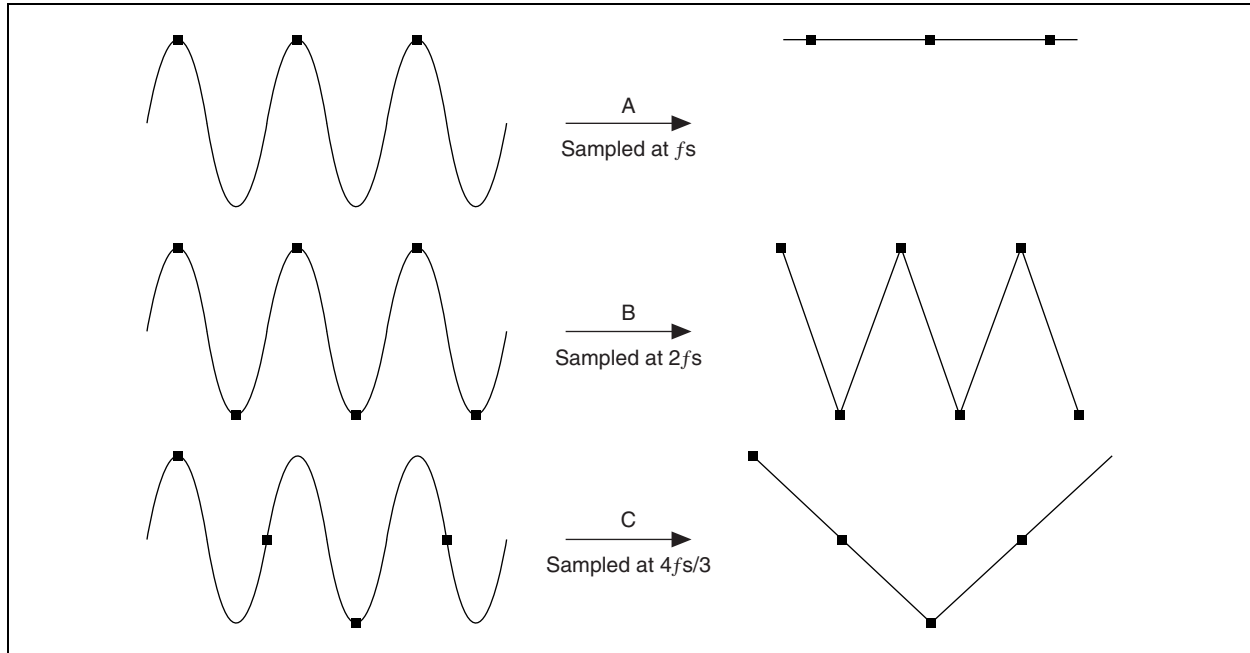
$$\text{Alias } F_2 = |100 - 70| = 30 \text{ Hz}$$

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

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

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. The following illustration shows the effects of various sampling rates.



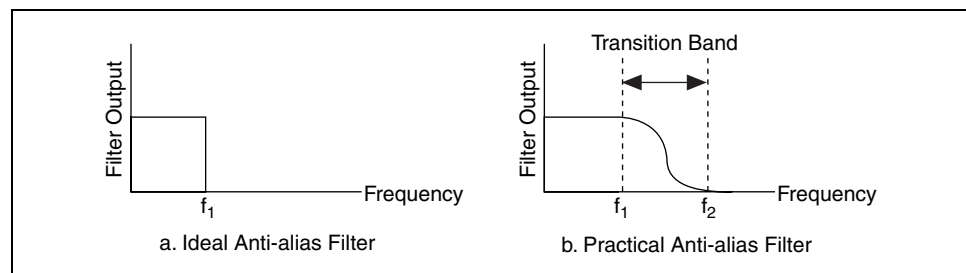
Example A, samples the sine wave of frequency f at the same frequency f_s . The acquired samples result in an alias at DC. However, if you increase the sampling rate to $2f_s$, the digitized waveform has the correct frequency or the same number of cycles as the original waveform but appears as a triangle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

The Nyquist theorem provides a starting point for the adequate sampling rate—greater than two times the highest frequency component in the signal. Unfortunately, this rate is often inadequate for practical purposes. Real-world signals often contain frequency components that lie above the Nyquist frequency and are erroneously aliased and added to the components of the signal that are sampled accurately, producing distorted sampled data. Therefore, for practical purposes, sampling is usually done at several times the maximum frequency—5 to 10 times is typical in industry.

B. Anti-aliasing Filters

You have seen that the sampling rate should be at least twice the maximum frequency of the signal that you are sampling. In other words, the maximum frequency of the input signal should be less than or equal to half of the sampling rate. But how do you ensure that this is definitely the case in practice? Even if you are sure that the signal being measured has an upper limit on its frequency, pickup from stray signals (such as the power line frequency or from local radio stations) could contain frequencies higher than the Nyquist frequency. These frequencies may then alias into the desired frequency range and thus give you erroneous results.

To be completely sure that the frequency content of the input signal is limited, a lowpass filter (a filter that passes low frequencies but attenuates the high frequencies) is added before the ADC. This filter is called an anti-alias filter because it prevents the aliasing components from being sampled by attenuating the higher frequencies (greater than Nyquist). Anti-aliasing filters are analog filters. The following illustration shows an ideal anti-alias filter.

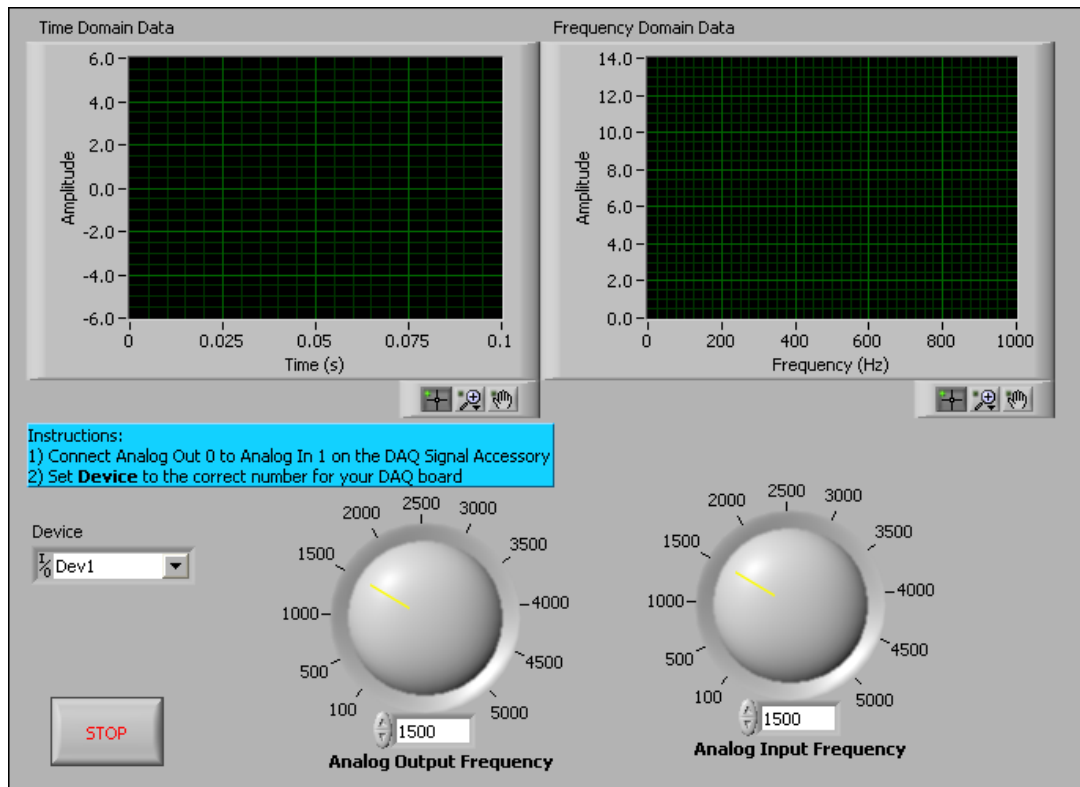


An ideal anti-aliasing filter passes all the desired input frequencies (below f_1) and cuts off all the undesired frequencies (above f_1). However, an ideal anti-aliasing filter is not physically possible. In practice, filters look as shown in illustration (b) above. Practical anti-aliasing filters pass all frequencies $< f_1$ and cut off all frequencies $> f_2$. The region between f_1 and f_2 is known as the transition band, which contains a gradual attenuation of the input frequencies. Although you want to pass only signals with frequencies $< f_1$, the signals in the transition band could still cause aliasing. Therefore, in practice, you should use a sampling frequency greater than two times the highest frequency in the transition band. Because this sampling frequency turns out to be more than two times the maximum input frequency (f_1), you might see that the sampling rate is more than twice the maximum input frequency.

Exercise 4-1 Sampling Rate and Aliasing

Objective: To demonstrate aliasing and the effects of sampling rate on an input signal.

1. Connect analog out 0 to analog in 1 on the DAQ Signal Accessory.
2. Open the Sampling Rate Example VI located on the C:\Exercises\LabVIEW DAQ directory. The following front panel appears.



This VI acquires a sine wave that is generated by the analog output circuitry on the DAQ device. The VI graphs both the time domain and frequency domain of the acquired signal.

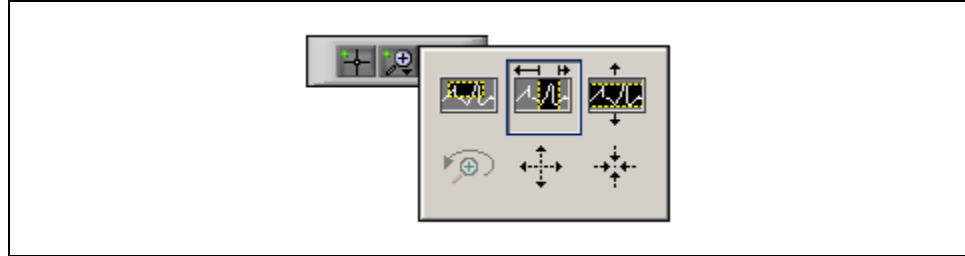
3. Set the front panel controls with the following values:
 - **Device:** Select the correct number for your DAQ device.
 - **Analog Output Frequency:** 500
 - **Analog Input Frequency:** 1500
4. Run the VI. The x coordinate of the peak you see on the Frequency plot represents the frequency of the sine wave the DAQ device generates.

Remember that the Nyquist Frequency (f_n) is $f_n = \frac{1}{2} f_s$.

With a sampling rate of 1,500 Hz, the Nyquist frequency is 750 Hz. This implies that the sampling rate is sufficient to measure a sine wave up to

750 Hz. When you run the VI, you see a peak at 500 Hz, which is the analog output frequency that the DAQ device generates.

5. Stop the VI. Click the **Zoom** button on the time-domain data graph and zoom in on the x-axis.



The data looks like a triangle wave. Because you are sampling three times faster than the analog output frequency, you are satisfying the Nyquist theorem, but you are not capturing the shape of the signal. Notice on the frequency-domain data graph that you have captured the correct frequency of the signal.

6. Run the VI. Increase the **Analog Input Frequency** to 5,000 Hz. The shape of the time domain signal looks like a smooth sine wave. Increasing the sampling rate 10 times faster than the signal you are trying to acquire more accurately represents the shape of the signal. In general, try to acquire a signal 5 to 10 times faster than the highest frequency in the signal you are trying to capture.
7. Set the **Analog Input Frequency** equal to 1,000 Hz. You are now sampling at $2fn$. When you sample at $2fn$, the time-domain signal looks like a triangle wave. You are accurately representing the frequency of the signal you are acquiring, but not the shape. If you increase the Analog Input Frequency just above 1,000 Hz, the frequency of the signal is also represented in the frequency-domain graph. As you slowly increase the Analog Input Frequency knob, you can see the frequency of the acquired signal. This implies that you must sample greater than $2fn$ to accurately represent the frequency domain of an acquired signal.
8. Reduce the **Analog Input Frequency** to 750 Hz. fn is equal to 375 Hz, which is lower than the acquired signal frequency. Although the time-domain data waveform appears sinusoidal in nature, the signal is aliased, which is indicated by the incorrect frequency displayed on the frequency domain plot. The alias frequency you see is determined by the following formula:

Alias freq. = | (closest integer multiple of the sampling freq. – signal freq.) |

Therefore, $|750 - 500| = 250$ Hz, which is what you see on the frequency domain graph.

The frequency domain graph displays the incorrect frequency because the frequency has been aliased between 0 and 375 Hz. The 500 Hz signal has been aliased to 250 Hz.

9. Close the VI. Do not save changes.

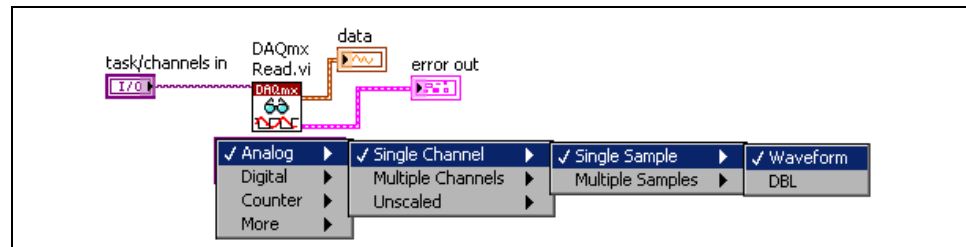


Tip When you select a sampling rate to obtain time-domain information such as the shape of the waveform, you must oversample at a rate of at least five times greater than the highest frequency component in the waveform. If you want to obtain only the frequency information, oversample at least two times greater than the highest frequency component in the waveform, according to the Nyquist Theorem.

End of Exercise 4-1

C. Using the DAQmx Read VI

The DAQmx Read VI located on the **DAQmx - Data Acquisition** reads samples from the task or channels you specify. The instances of this polymorphic VI specify what format of samples to return, whether to read a single sample or multiple samples at once, and whether to read from one or multiple channels. Use the pull-down menu to select an instance of the VI, as shown in the following figure.

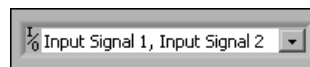
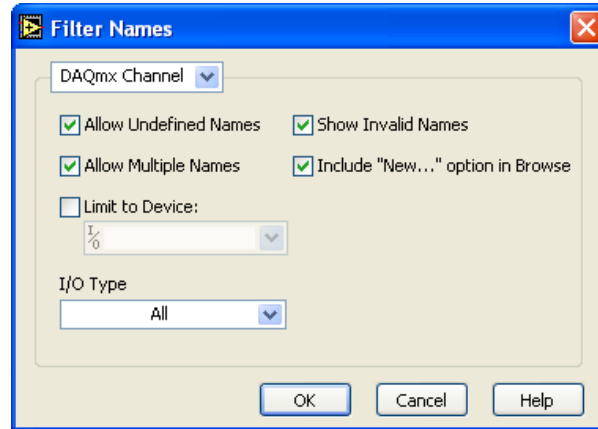


In the first selection menu you can choose from the following types of input:

- Analog
- Digital
- Counter
- More (Raw Data)

Use the second selection menu to determine the number of channels to read from or if the data is unscaled. Use the third selection menu to choose to read a single sample or multiple samples. If you select a single sample, use the fourth selection menu to select whether to return the data as a waveform or a double-precision, floating-point value. If you select multiple samples, use the fourth selection menu to select whether to return the data as a waveform or an array of double-precision, floating point values.

When you are addressing analog input or analog output channels, you might want to address more than one channel at a time. If these channels have the same type of timing and triggering, group the channels into a task. Otherwise, use the I/O Name Filtering tool on the shortcut menu of the NI-DAQmx task or channel control/constant and select **Allow Multiple Names** (selected by default). Separate the channel names with a comma. You cannot address multiple tasks at a time.



Waveform Data Type

The waveform data type is a cluster that consists of the following elements:

- **Y**—A 1D array of numeric data points, which can be a single point or a waveform depending on the operation. The representation of the 1D array is DBL.
- **t_0** —A scalar value that represents the time, according to the system clock, when the first point in the Y array was acquired. This element is also called the initial time or the timestamp.
- **Δt** —A scalar value that represents the time between data points in the Y array.
- **Attributes**—A string that allows you to bundle other information along with the waveform, such as the device number or channel number.

The waveform data type has many benefits over the conventional scaled array.

- **The presence of t_0** —Before the waveform data type existed, you could not determine when data was acquired. The waveform data type automatically returns the time of day and the date in the t_0 element, which gives you a real-world acquisition time for the data.
- **Easier Graphing**—The waveform data type simplifies graphing the data. Previous versions of LabVIEW required you to bundle the value of the initial point (x_0) and the time between points (Δx) with the data (Y array). The waveform data type contains these elements, so all you have to do is wire the waveform data type to the graph.

- **Easier Multiple Plot Graphing**—The waveform data type simplifies multiple plot graphs. Previous versions of LabVIEW required you to bundle the x_0 , Δx , and Y array for each plot, then build an array to create a multiple plot graph. Using a waveform data type, you wire a 1D array of waveforms to the graph for a multiple plot. If you acquire data on multiple channels with an analog input VI, the VI returns a 1D array, which you can wire directly to the graph.

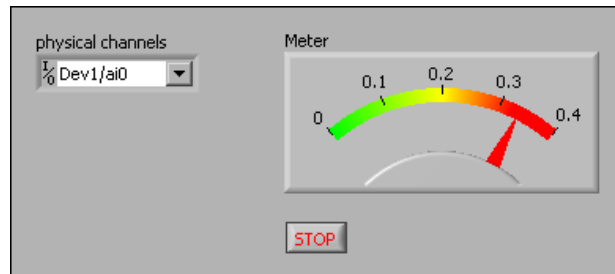
Exercise 4-2 Voltmeter VI

Objective: To acquire an analog signal using a DAQ device.

Complete the following steps to build a VI that measures the voltage that the temperature sensor on the DAQ Signal Accessory returns. The temperature sensor returns a voltage proportional to the temperature. The sensor is hardwired to channel 0 of the DAQ device.

Front Panel

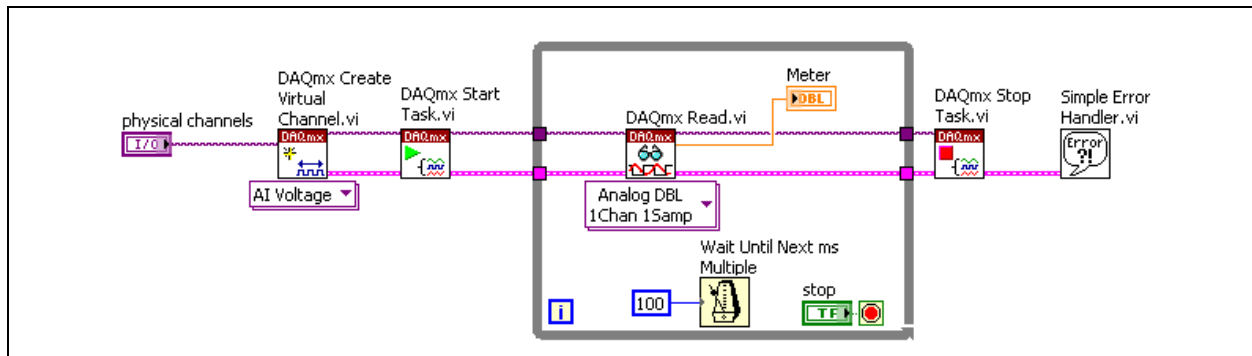
1. Open a blank VI and build the following front panel.



Configure the meter scale for 0.0 to 0.4. Use the Labeling tool to double-click 0.0 and type 0.4. You might need to enlarge the meter to display the scale.

Block Diagram

2. Build the following block diagram.



- a. Place the DAQmx Create Virtual Channel VI, located on the **Functions»All Functions»NI Measurements»DAQmx - Data Acquisition** palette, on the block diagram. This VI creates a virtual channel of the type you specify in the instance pull-down menu. Select the **AI Voltage** instance from the pull-down menu.



b. Place the DAQmx Start Task VI, located on the **Functions»All Functions»NI Measurements»DAQmx - Data Acquisition** palette, on the block diagram. This VI starts the measurement task.



c. Place a While Loop, located on the **Functions»All Functions»Structures** palette, on the block diagram. The While Loop repeats the subdiagram inside it until the conditional terminal receives a particular Boolean value.



d. Place the DAQmx Read VI, located on the **Functions»All Functions»NI Measurements»DAQmx - Data Acquisition** palette, on the block diagram. This VI performs the measurement reading you specify in the instance pull-down menu. Select the **Analog»Single Channel»Single Sample»DBL** instance. This instance returns a single analog sample of double-precision, floating-point numeric data from a single channel.



e. Place the Wait Until Next ms Multiple function, located on the **Functions»All Functions»Time & Dialog** palette, on the block diagram. This function causes the loop to execute every 100 ms.



f. Place the DAQmx Stop Task VI, located on the **Functions»All Functions»NI Measurements»DAQmx - Data Acquisition** palette, on the block diagram. This VI stops the measurement task.



g. Place the Simple Error Handler VI, located on the **Functions»All Functions»Time & Dialog** palette, on the block diagram. In the event of an error, this VI displays a dialog box with information about the error and where it occurred.

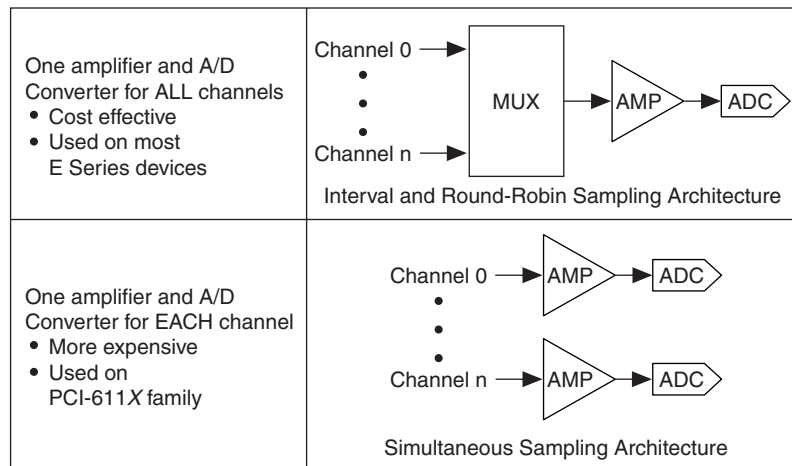
3. Save the VI as `Voltmeter.vi` in the `C:\Exercises\LabVIEW DAQ` directory.
4. Display the front panel and set the physical channel as `DevX/ai0`, where X is the device number of your DAQ device as set in MAX.
5. Run the VI. The meter displays the voltage the temperature sensor outputs. Place your finger on the temperature sensor and notice that the voltage increases.
6. Stop the VI.
7. Place a **DAQmx Global Channel** control, located on the **Controls»All Controls»I/O»DAQmx Name Controls** palette, on the front panel.
8. Return to the block diagram and delete the DAQmx Create Virtual Channel VI and the **physical channels** control.
9. Wire the **DAQmx Global Channel** control to the **task/channels in** input of the DAQmx Start Task VI.

10. Return to the front panel and select the channel `Temperature Sensor`. Run the VI. The temperature displays in the meter. The temperature values are 100 times greater than the voltage values because the NI-DAQmx channel uses a custom scale to multiply the voltage by one hundred. Change the meter scale to see the correct values.
11. Save and close the VI.

End of Exercise 4-2

D. DAQ Device Architectures

The number and arrangement of the components on the device depends on the DAQ device you use. The architecture of the device affects how you sample a signal. National Instruments DAQ devices that perform analog input can have one of two main architectures, as shown in the following table.



The interval and round-robin sampling architecture consists of one multiplexer, one instrumentation amplifier, and one ADC. In this layout, all the input channels must share one ADC. Using only one ADC makes this architecture very cost effective, so it is used on most E Series devices. The simultaneous sampling architecture consists of an instrumentation amplifier and an ADC for each channel. This architecture is used on the PCI-611X family of devices. Although this architecture is more expensive than using one ADC for all channels, it allows you to perform simultaneous sampling.

Sampling Terminology

Samples per Channel per Second—The number of samples to acquire for each second.

Sample Clock—A pulse train used to start sample acquisition. Each time the sample clock produces a pulse, one sample per channel is acquired.

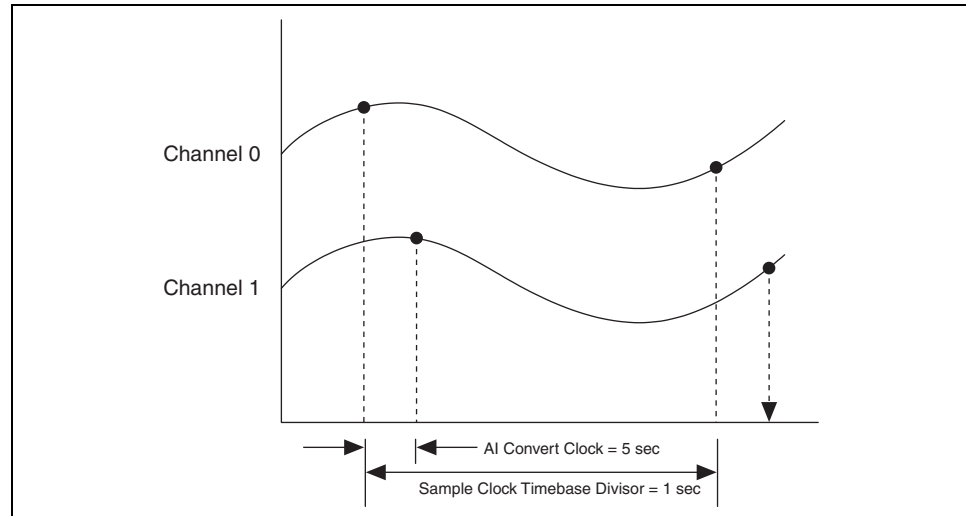
AI Convert Clock—A pulse train used to trigger an A/D conversion.

Sample Duration—The time it takes to complete one set of samples. Use the following formula to calculate the sample duration:

$$\text{Sample Duration} = (\# \text{ of channels} - 1) \times \text{AI Convert Clock}$$

Interval Sampling

In sampling a signal, you can choose from interval sampling, round-robin sampling, or simultaneous sampling. The following illustration shows an example of interval sampling.

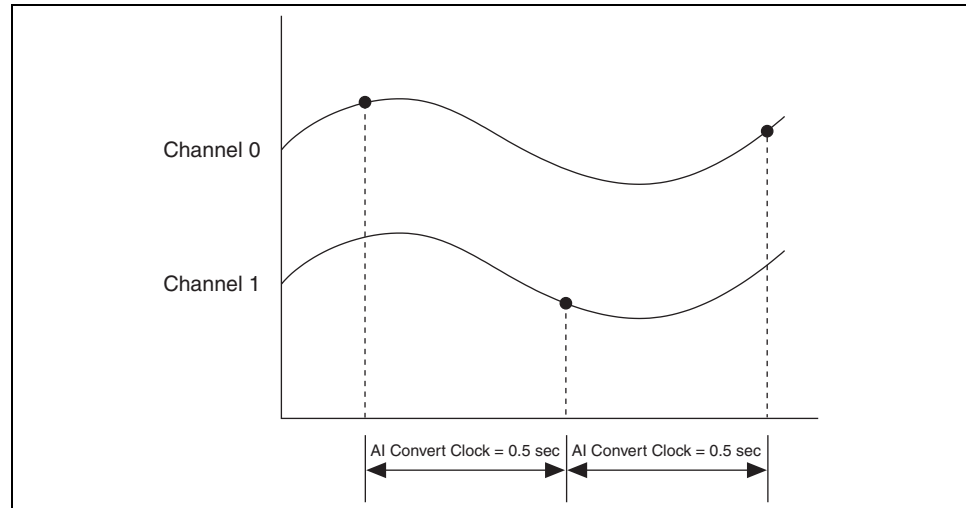


The most common method, interval sampling, shares one ADC between all the channels on the device. This architecture is found on most E Series devices. Interval sampling uses a sample clock and the AI Convert Clock to control the multiplexer (MUX). To understand how these two clocks interact, assume you are acquiring data on two channels. When the sample clock signals the start of an acquisition, the MUX connects the first channel to the ADC, and the AI Convert Clock pulses once. When the AI Convert Clock pulses, the ADC acquires one point from the first channel. Before the AI Convert Clock pulses again, the MUX connects the second channel to the ADC. When the AI Convert Clock pulses again, the ADC takes one point from the second channel. When the sample duration has elapsed, the sample clock pulses again, and the cycle repeats. The sample clock determines how often the device takes a sample of all the channels. The AI Convert Clock actually takes the samples. Because interval sampling uses a sample clock and the AI Convert Clock, the device is able to sample the channels in a short period of time.

In the previous illustration, the device takes a sample from each channel every second, but the lag between samples is only 5 μ s, as determined by the period of the AI Convert Clock. For the cost-effectiveness of having only one ADC, you can achieve near-simultaneous sampling.

Round-Robin Sampling

Round-robin sampling also uses one ADC for all channels. The difference between this method and interval sampling is that round-robin sampling does not use a scan clock. The channel clock starts the scan and determines the time between samples. The following illustration shows an example of round-robin sampling.

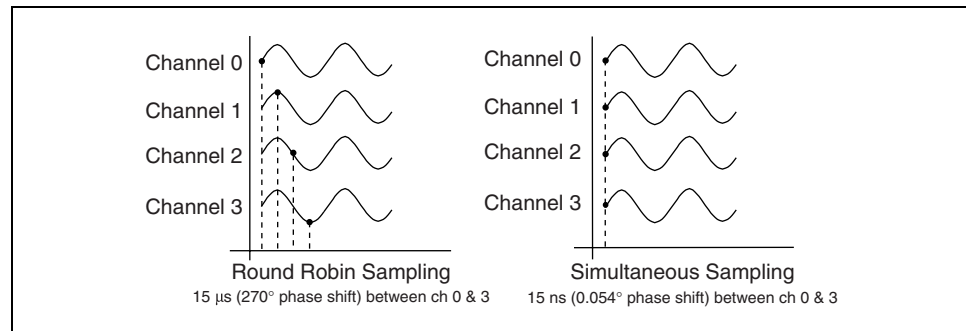


In the interval sampling example, the device started a sample on each channel every second and took samples on two different channels. The round-robin sampling example in the previous illustration uses the same conditions. However, it has only one clock, so all the points must be evenly spaced. The only way to evenly space the points and meet the one sample per second and two samples per channel per second criteria is to use a AI Convert Clock rate of two samples per second. The difference between this example and the interval sampling example is that the sample duration is now 0.5 seconds instead of 5 μ s.

With interval sampling, points on separate channels are not taken far apart in time. With round-robin sampling, the samples are farther apart in time. While round-robin sampling is simpler because it uses only one clock, you can use it only when the time relationship between signals is not important. Round-robin sampling is found on legacy devices.

Simultaneous Sampling

If the time relationship between the signals is important, you could use interval sampling, but sometimes interval scanning does not preserve the time relationship between signals to a narrow enough tolerance. In this case, you should use simultaneous sampling, as shown in the following illustration.



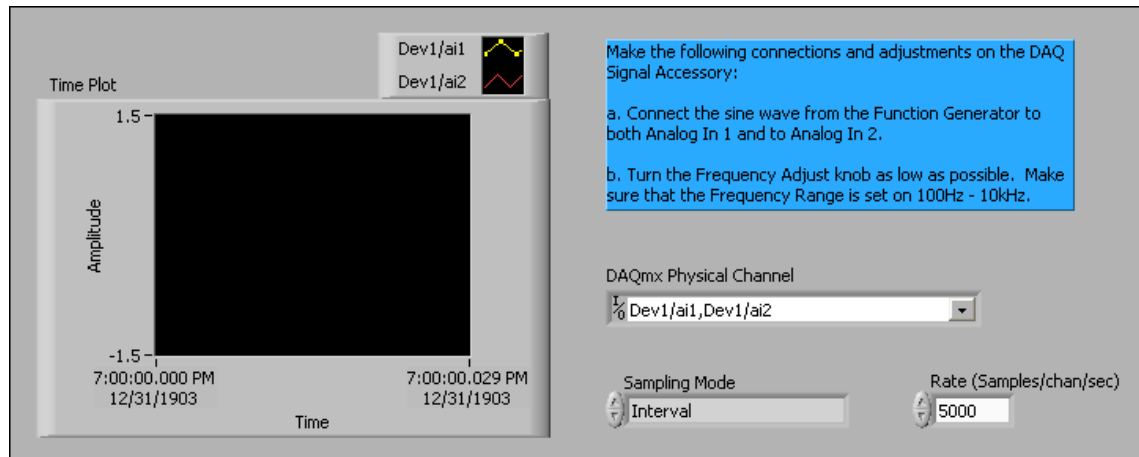
Simultaneous sampling uses one ADC for each channel so you can sample all channels at the same time. While this requires a more expensive architecture than interval scanning, it eliminates the lag between channels caused by having to share the ADC between all channels. Because simultaneous sampling samples every channel at the same time, you only need a sample clock to determine the sampling rate.

Compare all three types of sampling by determining the phase shift that occurs when you sample four 50 kHz signals at a rate of 200 kHz. With round-robin sampling, all the samples must be evenly spaced, which causes a 15 μ s delay between the time of the sample on channel 0 to the time of the sample on channel 3. This corresponds to a 270 degree phase shift. With interval scanning, assume you have a 5 μ s interchannel delay. Again, you experience a 15 μ s delay between channel 0 and channel 3. With simultaneous sampling, you experience only a three nanosecond delay between channel 0 and channel 3. This delay corresponds to a 0.054 degree phase shift. Simultaneous sampling provides a great advantage in preserving the time relationship between signals, albeit at a higher cost. The PCI-611X family can perform simultaneous sampling.

Exercise 4-3 Interval Sampling vs. Round-Robin Sampling (Optional)

Objective: To observe the differences between round-robin sampling and interval sampling.

1. Make the following connections and adjustments on the DAQ Signal Accessory:
 - a. Connect the sine wave from the function generator to both analog in 1 and analog in 2 on the DAQ Signal Accessory.
 - b. Turn the Frequency Adjust knob on the DAQ Signal Accessory as low as possible. Make sure that the Frequency Range is set on 100 Hz – 10 kHz.
2. Open the Interval vs. Round-Robin Sampling Example VI located on the C:\Exercises\LabVIEW DAQ directory. The front panel is shown in the following figure.



This VI acquires waveforms from two analog input channels and plots them on the graph.

3. Set the controls with the following values:
 - **DAQmx Physical Channel:** DevX/ai1, DevX/ai2
where x is the number of your DAQ device, as set in MAX.
 - **Rate:** 5000
 - **Sampling Mode:** Round-Robin

4. Run the VI. Notice the waveforms are offset even though the two channels are sampling the same signal. Remember two facts about round-robin sampling:
 - Multiple channels cannot be sampled simultaneously. The MUX feeding the ADC switches between analog input channels. Because the channel clock controls when the mux is switched, a time skew occurs between any two successive channel samples. Thus, round-robin sampling is appropriate for applications in which the time relationship between each sampled channel is not important.
 - The maximum sampling rate per channel is inversely proportional to the number of channels sampled. For example, a DAQ device sampling at 700 kHz on 10 channels effectively samples each channel only at 70 kHz.
5. Change the **Sampling Mode** to **Interval Sampling** and run the VI again. Notice the signals appear on top of each other as if the two channels were sampled simultaneously. Round-robin sampling has only the channel clock, which is responsible for the sampling rate and the time between channels. With interval sampling, the sample clock is responsible for the samples per channel per second rate only. Another clock, the AI convert clock, controls the time between samples. Having two clocks allows you to set the AI convert clock as fast as possible, which creates the effect of simultaneous sampling.
6. Close the VI. Do not save changes.

End of Exercise 4-3

E. Multiple-Point (Buffered) Analog Input

To acquire multiple points at the same time, select an instance of the DAQmx Read VI that reads multiple samples. Use the DAQmx Read VI in combination with the DAQmx Timing VI, the DAQmx Start Task VI, and the DAQmx Stop Task VI to create a hardware-timed, buffered acquisition VI.

- **Hardware-Timed Acquisition**—A hardware signal such as a sample clock or AI Convert Clock controls the rate of acquisition. A hardware clock is much faster than a software loop, so you can sample a higher range of frequencies without aliasing the signal. A hardware clock also is more accurate than a software loop. A software loop rate can be affected by a variety of actions, such as the opening of another program on the computer, but a hardware clock stays consistent.
- **Buffered Acquisition**—Acquires multiple points with one call to the device. The points are transferred from the device to an intermediate memory buffer before LabVIEW reads them.

DAQmx Timing VI

The DAQmx Timing VI configures the sample rate, the number of samples to acquire or generate and creates a buffer when needed. The instances of this polymorphic VI correspond to the type of timing to use on the task. The available timing options are Sample Clock, Handshaking, Implicit, and Use Waveform.

For analog input, select the Sample Clock instance in the pull-down menu of the DAQmx Timing VI. This instance of the VI includes the following parameters:

- **sample mode**—Specifies if the task executes continuously or for a finite duration.
- **samples per channel**—Specifies the number of samples to input or output if **sample mode** is **Finite Samples**. This value specifies the size of the intermediate memory buffer that stores the data as it is transferred from the DAQ device to LabVIEW.
- **rate**—Specifies the sampling rate in samples per channel per second. If you use an external source for the Sample Clock, set this input to the maximum expected rate of that clock.
- **source**—Specifies the source terminal of the Sample Clock. Leave this input unwired to use the default onboard clock of the device.

- **active edge**—Specifies on which edge of the clock to measure or generate samples. Select the rising or falling edge of the sample clock.
- **task/channels in**—Specifies the name of the task or a list of virtual channels the operation applies to. If you provide a list of channels, NI-DAQmx creates a task automatically.

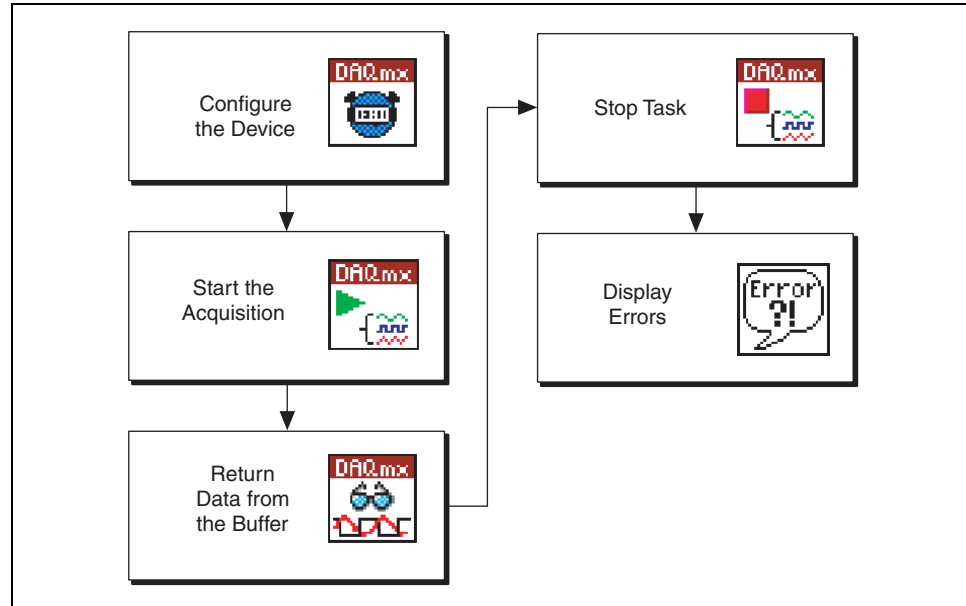
The Handshaking instance of the DAQmx Timing VI determines the number of digital samples to acquire or generate using digital handshaking between the device and a peripheral device. Refer to Lesson 8, *Digital I/O*, of this manual for more information about the Handshaking instance of the DAQmx Timing VI.

The Implicit instance of the DAQmx Timing VI sets only the number of samples to acquire or generate without specifying the timing. Typically, you should use this instance when the task does not require sample timing, such as tasks that utilize counters for buffered frequency measurement, buffered period measurement, or pulse train generation.

The Use Waveform instance of the DAQmx Timing VI uses the **dt** component of the **waveform** input to determine the sample clock rate. **dt** is the time in seconds between samples. If **sample mode** is **Finite Samples**, NI-DAQmx generates the number of samples in the waveform. This VI does not actually output any samples. You must wire the same waveform to the DAQmx Write VI to produce the samples. Refer to Lesson 7, *Analog Output*, of this manual for more information about the Use Waveform instance of the DAQmx Timing VI.

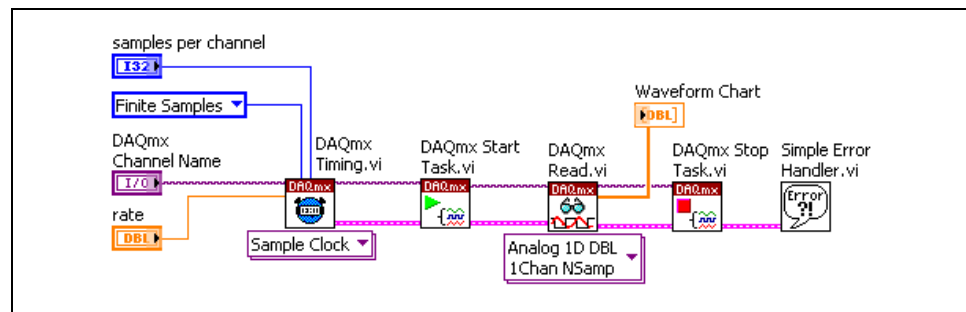
Buffered Acquisition Flowchart

The following flowchart shows a basic buffered acquisition. A buffered acquisition acquires a set number of points at a specified rate. Use the DAQmx Timing VI to configure the timing and buffer for the device. Use the DAQmx Start Task VI to start the acquisition. The DAQmx Read VI waits until all the samples on each channel are available before returning the data and moving on. The DAQmx Stop Task VI stops the task and frees the resources assigned to the device. The Error Handler VI displays any errors that occurred during the process.



Buffered Acquisition Example

The following example shows how to create a buffered acquisition VI. The DAQmx Timing VI sets up the task/channel, timing, and samples per channel (buffer size). Next, the DAQmx Start Task VI starts the acquisition. The program then waits at the DAQmx Read VI until the buffer is full. When the buffer is full, the DAQmx Read VI returns the data from the buffer, the DAQmx Stop Task VI stops the acquisition, and the Simple Error Handler VI displays any errors.

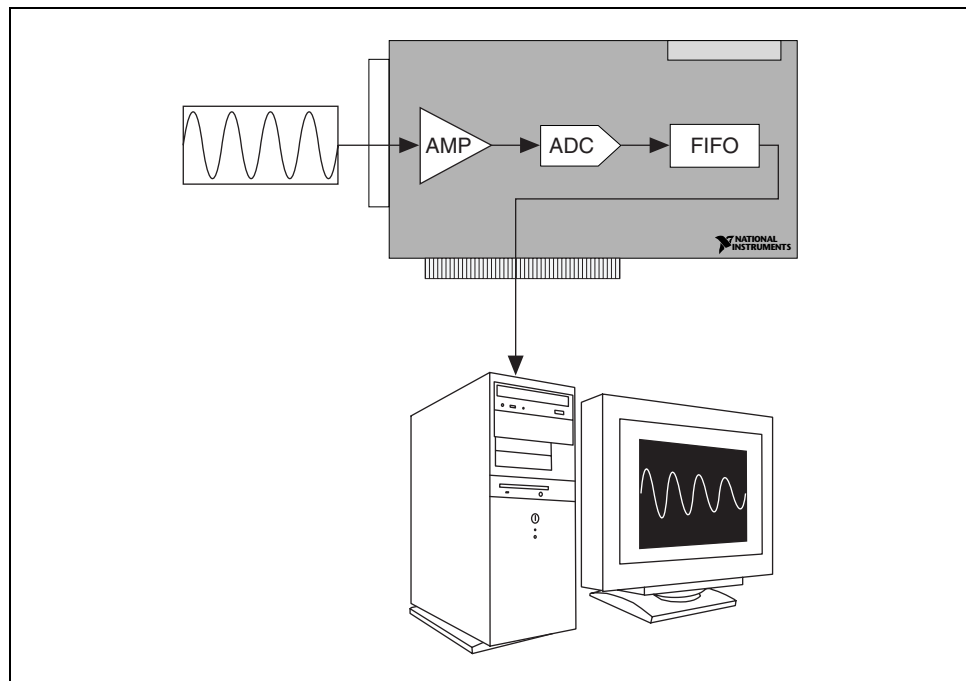


Because the **number of samples per channel** input for the DAQmx Read VI is unwired, NI-DAQmx automatically determines how many samples to read based on the configuration you set in the DAQmx Timing VI. NI-DAQmx automatically determines this value and the **number of samples per channel** input is set to -1 . The DAQmx Read VI returns a 2D array that can be directly wired to a waveform graph. The array does not include timing information, unlike the waveform data type.

Always wire the error clusters from one VI to another. If **error in** detects an error in the DAQmx Start Task VI, DAQmx Read VI, or DAQmx Stop Task VI, the VI returns the error information in **error out** and does not continue to run. For example, assume an error occurs in the DAQmx Start Timing VI. The DAQmx Start Timing VI stops executing and passes the error information to the DAQmx Start Task VI. The DAQmx Start Task VI does not execute—it passes the error to the next VI. The error information passes through each VI to the Error Handler VI for display.

What's Really Happening?

To understand what happens when you perform a buffered acquisition, examine a buffered acquisition on a lower level, as shown in the following illustration.



You already know that when you acquire an analog signal, it passes through the instrumentation amplifier to the ADC. However, you might not know what happens to the signal after that. The signal passes to an onboard First In First Out (FIFO) buffer which stores the data until it can be transferred from the device to the computer. The data is then transferred from the device to a PC buffer through Direct Memory Access (DMA) or Interrupt Request (IRQ).

The PC buffer is a memory location that stores the data after it has left the device. The **number of samples per channel** input of the DAQmx Timing VI (or the **buffer size** input of the DAQmx Configure Input Buffer VI) configures the PC buffer, which stores the data until DAQmx Read VI is ready to retrieve it. The DAQmx Read VI then transfers the data to a LabVIEW buffer, which can then be displayed on the front panel. The LabVIEW buffer can place the data in a waveform graph, an array, or a waveform data type, depending on the DAQmx Read VI instance and how you wire the output of the DAQmx Read VI.

Buffer Transfer

The transfer of data between the PC buffer and the LabVIEW buffer is important in analog input operations. The **number of samples per channel** input of the DAQmx Timing VI allocates the PC buffer. When you perform a buffered acquisition, the acquisition begins when you call the DAQmx Start Task VI. After the acquisition begins, the PC buffer starts filling with data. A buffered acquisition fills the PC buffer until it is full. The rate at which the buffer fills is determined by the rate that you set in the DAQmx Timing VI. When the buffer is full, the DAQmx Read VI transfers the data from the PC buffer to the LabVIEW buffer. The DAQmx Read VI removes all the data at one time in a buffered acquisition.

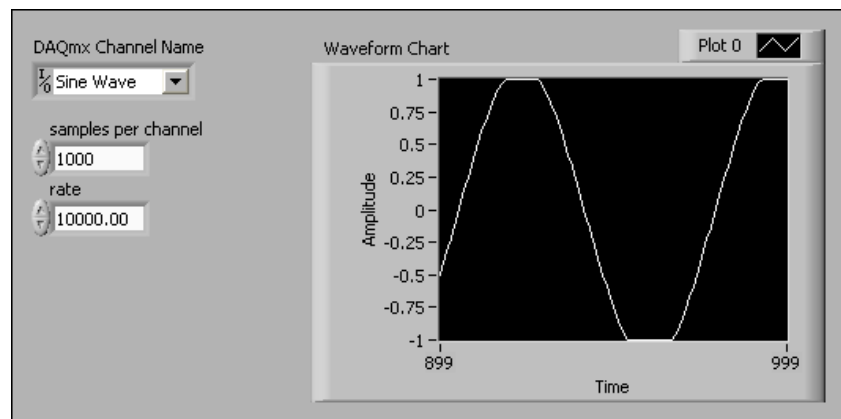
Exercise 4-4 Finite Buffered Acquisition

Objective: To acquire an array of data using a finite buffered configuration.

With a finite buffered acquisition, LabVIEW specifies how many points to acquire and at what rate to acquire them. Timing then becomes the responsibility of the DAQ device. In a buffered acquisition, the DAQ device controls all aspects of the acquisition. In contrast, with a software-timed acquisition, the computer is solely responsible for managing the acquisition, which can be problematic if the computer suddenly cannot give priority to the data acquisition process.

Front Panel

1. Open a blank VI and build the following front panel.



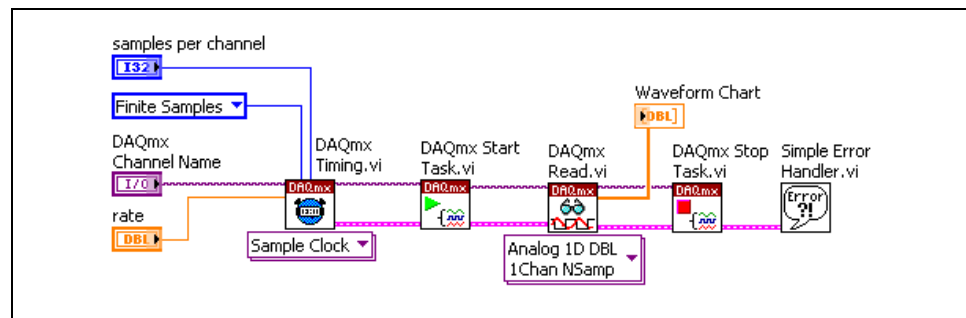
You can create most of the front panel controls shown above from the block diagram by right-clicking the appropriate terminals of the DAQ VIs and selecting **Create»Control**.

In this exercise, you acquire data from one channel on the DAQ Signal Accessory and display the data on the graph. Set the **samples per channel** (buffer size) to 1000, and the **rate** to 10000. Set the **DAQmx Global Channel** control to *Sine Wave*.

2. Connect the sine wave output to analog input CH1 on the DAQ Signal Accessory.

Block Diagram

- Build the following block diagram.



- Place the DAQmx Timing VI, located on the **Functions»All Functions»NI Measurements»DAQmx - Data Acquisition** palette, on the block diagram. This VI configures the sample timing and buffer size (samples per channel) of the task. Select the **Sample Clock** instance from the pull-down menu to use the internal clock on the DAQ device.



- Place the DAQmx Start Task VI, located on the **Functions»All Functions»NI Measurements»DAQmx - Data Acquisition** palette, on the block diagram. This VI starts the measurement task.



- Place the DAQmx Read VI, located on the **Functions»All Functions»NI Measurements»DAQmx - Data Acquisition** palette, on the block diagram. This VI performs the type of measurement reading you specify in the instance pull-down menu. Select the **Analog»Single Channel»Multiple Samples»1D DBL** instance to return a 1D array of double-precision, floating-point numeric data from a single channel.



- Place the DAQmx Stop Task VI, located on the **Functions»All Functions»NI Measurements»DAQmx - Data Acquisition** palette, on the block diagram. This VI stops the measurement task.

- Save the VI as `Buffered Acquisition.vi` in the `C:\Exercises\LabVIEW DAQ` directory.
- Return to the front panel and run the VI. A sine wave should plot on the waveform graph.
- Close the VI.

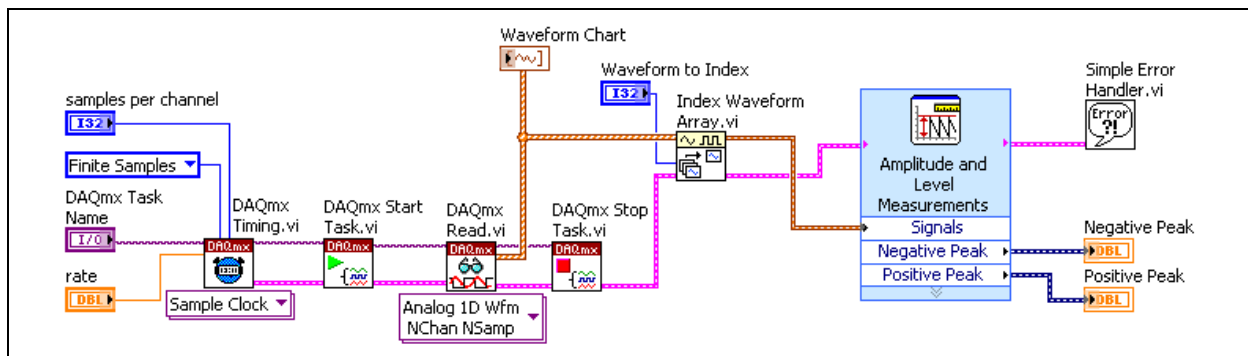
End of Exercise 4-4

Exercise 4-5 Buffered Acquisition with Waveform Analysis

Objective: To acquire waveforms using a buffered configuration and to analyze this data for the maximum and minimum values.

The Waveform Min Max VI allows you to see the minimum and maximum values of the sine wave. These values help you determine if the generator is operating within its specifications.

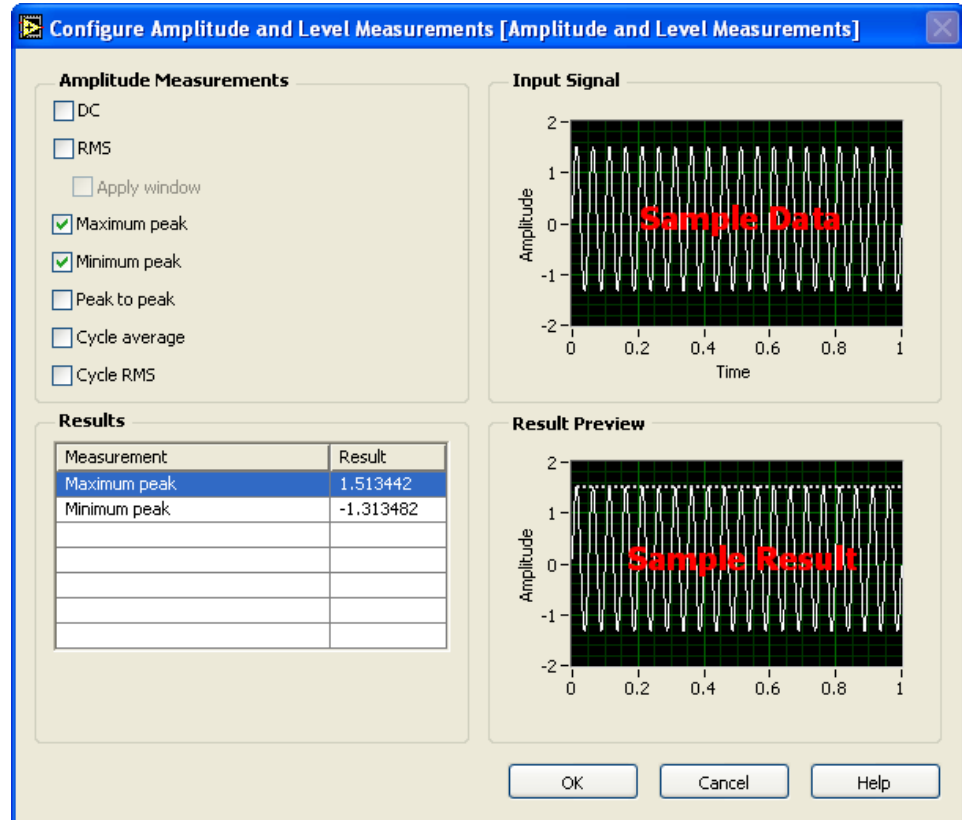
1. Open the Buffered Acquisition VI located in the C:\Exercises\LabVIEW DAQ directory that you completed in Exercise 4-4.
2. Select **File»Save As** and save the VI as Buffered Acquisition (min max) .vi in the C:\Exercises\LabVIEW DAQ directory.
3. On the front panel, replace the **DAQmx Global Channel** control with a **DAQmx Task Name** control by right-clicking the channel control and selecting **Replace»I/O»DAQmx Name Controls»DAQmx Task Name** from the shortcut menu.
4. Modify the block diagram as shown in the following figure.



- a. Place the Index Waveform Array function, located on the **Functions»All Functions»Waveform»Waveform Operations** palette, on the block diagram. This function selects a waveform from an array of waveforms.



- b. Place the Amplitude and Level Measurements Express VI, located on the **Functions»Signal Analysis** palette, on the block diagram. In this exercise, the Amplitude and Level Measurements Express VI determines the maximum and minimum values in a signal.
 - In the **Configure Amplitude and Level Measurements** dialog box that appears, place checkmarks in the **Maximum peak** and **Minimum peak** checkboxes.
 - Click the **OK** button to apply the changes and close the dialog box.



5. To create the Negative Peak and Positive Peak front panel indicators, right-click the corresponding outputs on the Amplitude and Level Measurements Express VI and select **Create»Numeric Indicator** from the shortcut menu.
6. Set the front panel controls with the following values:
 - **DAQmx Channel Name:** My Voltage Task
 - **Samples per Channel:** 1000
 - **Rate:** 10000
7. Connect the sine wave output to analog input CH1 and the square wave output to analog input CH2 on the DAQ Signal Accessory.
8. Run the VI. Determine the maximum and minimum values of the square wave by changing the **Waveform to Index** to 2.



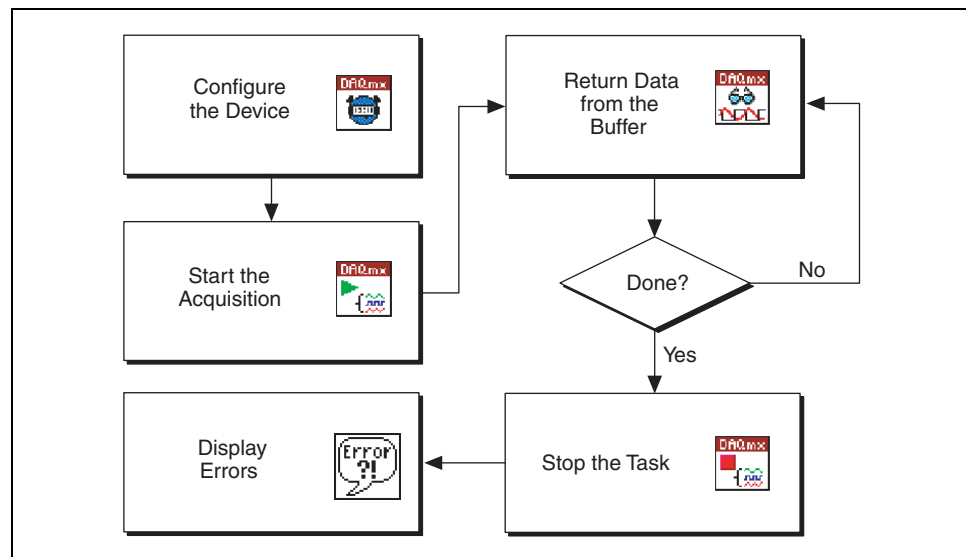
Note Because the waveforms are indexed starting at zero, the waveform indexes are 0, 1, and 2.

9. Double-click the Amplitude and Level Measurements Express VI and select different amplitude values to determine. Create indicators for these values and run the VI.
10. Save and close the VI.

End of Exercise 4-5

F. Continuous Acquisition Flowchart

The main difference between a finite buffered acquisition and continuous buffered acquisition is the number of points that you acquire. With a finite buffered acquisition, you acquire a set number of points. With a continuous buffered acquisition, you can acquire data indefinitely. The following flowchart shows a continuous buffered acquisition.

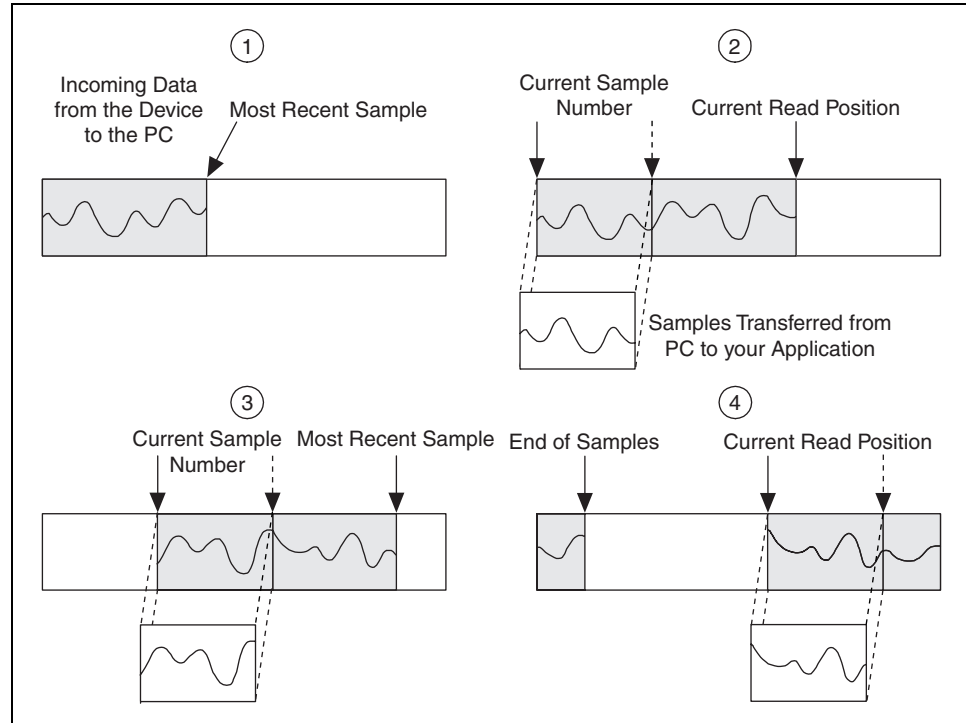


The first three steps of this flowchart are identical to the first three steps of the buffered acquisition flowchart. Configure a device with the DAQmx Timing VI, start the acquisition with the DAQmx Start Task VI, and prepare to read the data with the DAQmx Read VI. Because you acquire data continuously, you need to read data continuously. Place the DAQmx Read VI in a loop. The loop completes when an error occurs or when you stop the loop from the front panel. Every time the loop runs, the DAQmx Read VI returns data. When the loop ends, the DAQmx Stop Task VI stops the tasks and unassigns the resources. The Simple Error Handler VI displays any errors that occurred during the process.

Continuous Buffered Acquisition

The following block diagram of a continuous buffered acquisition VI is similar to a buffered acquisition with the following changes:

- The DAQmx Read VI is inside a While Loop.
- The **number of samples per channel** input is user-specified. With finite acquisition, NI-DAQmx automatically determines how many samples to read. If you leave the **number of samples per channel** input unwired or set to -1 , NI-DAQmx reads the total number of samples available in the buffer.



A circular buffer is similar to a regular buffer, but when you get to the end of a circular buffer, instead of stopping, you start over at the beginning. Start with the PC buffer that was allocated by the **samples per channel** input of the DAQmx Timing VI. When the DAQmx Start Task VI starts the acquisition, the PC buffer starts to fill with data. The acquisition occurs inside the While Loop.

Assume that you set the **number of samples per channel to read** to between one-fourth and one-half of the PC buffer size. When the number of samples per channel in the PC buffer is equal to the **number of samples per channel to read**, the DAQmx Read VI transfers that number of samples per channel from the PC buffer to the LabVIEW buffer. The DAQmx Read VI sets a flag called the current sample position, so it can continue reading where it left off.

Meanwhile, the PC buffer continues to fill with data. The DAQmx Read VI continues transferring data from the PC buffer to the LabVIEW buffer while the PC buffer fills. When the end of data mark reaches the end of the PC buffer, the new data is written at the beginning of the buffer. The difference between the end of samples mark and the current sample position is equal to the number of available samples per channel (backlog). LabVIEW must read the data from the buffer fast enough to prevent the end of samples mark from catching up to the current sample position, otherwise the new data overwrites the old data, and LabVIEW generates an error.

Overwrite Error

The most common error you might encounter when performing a circular buffered acquisition is the overwrite error. The overwrite error occurs when the end of samples mark catches up to the current sample position and you overwrite data. The problem occurs when LabVIEW does not read data from the PC buffer quickly enough. Several options exist to help avoid the error, but not all options might apply to the situation, and some work better than others.

- Increase the number of samples per channel (buffer size) with the DAQmx Timing VI. Increasing the buffer size does not solve the problem if you are not emptying the buffer fast enough. Remember the guideline for setting the samples per channel to read at one-fourth to one-half of the buffer size. Increasing the buffer size only works if it makes this guideline true.
- Empty the buffer more quickly by increasing the number of samples per channel to read. Do not set the number of samples per channel to read too high because you will wait in the DAQmx Read VI for the number of samples per channel in the buffer to equal the number of samples per channel to read. The time spent waiting for samples to fill the buffer could be spent emptying the buffer.
- Decrease the samples per channel rate with the DAQmx Timing VI. This setting slows down the rate that data is being sent to the buffer, but it might not be an option if you want a certain sample rate.
- Avoid slowing down the loop with unnecessary analysis.

Overflow Error

Another error you might encounter with a continuous buffered acquisition involves overflowing the FIFO buffer on the device. Overflow error is not as common as overwriting the PC buffer, and it is not as easy to correct. The problem occurs when the FIFO buffer does not empty fast enough. The FIFO buffer relies on either DMA or IRQ to transfer the data from the FIFO buffer to the PC buffer. When the FIFO buffer does not empty fast enough, you have only a few options to prevent the error.

- Make sure you are using DMA to transfer the data, if DMA is available. DMA is faster than IRQ and can improve performance significantly. For more information about using DMA, refer to the *NI-DAQmx Help* and the DAQmx Channel Property Node, Data Transfer Mechanism property.
- Decrease the samples per channel rate in the DAQmx Timing VI.

- Purchase a device with a larger FIFO buffer. However, this option might delay the problem instead of solving it.
- Purchase a computer with a faster bus to expedite the data transfer from the FIFO buffer to the PC buffer. Overflow is caused by the system not transferring the data off the device fast enough. A computer with a faster bus can transfer the data from the FIFO buffer faster.

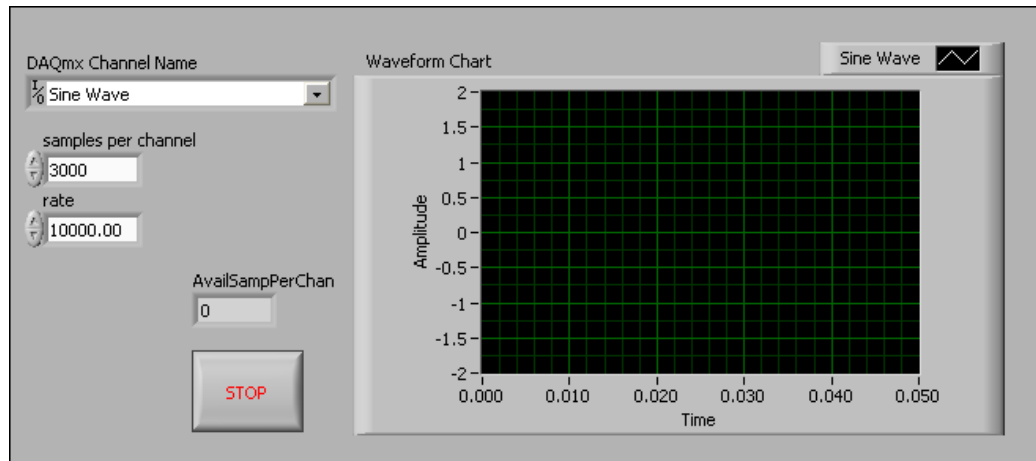
Exercise 4-6 Continuous Buffered Acquisition VI

Objective: To continuously acquire data from a DAQ device and log the data to a file.

Complete the following steps to build a VI that performs a continuous acquisition operation and plots the most recently acquired data on a chart.

Front Panel

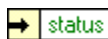
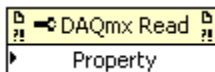
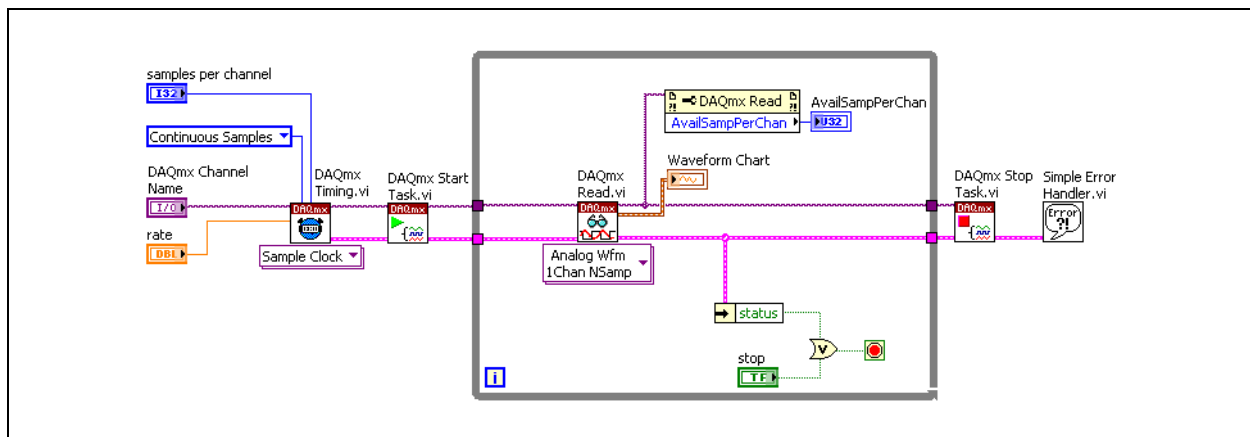
1. Open a blank VI and build the following front panel.



- a. You can create most of the front panel controls on the block diagram by right-clicking the appropriate terminals of the DAQmx VIs and selecting **Create»Control** from the shortcut menu.
 - b. Right-click the waveform chart and select **Properties**. Click the **Format and Precision** tab. Format the x-axis to floating point values with 3 digits of precision.
 - c. Click the **Scales** tab. For the y-axis, deselect **Autoscale Y** and enter the minimum and maximum values as -2 and 2 , respectively. Repeat this step for the x-axis, but set the minimum and maximum values to 0.00 and 0.05 , respectively.
2. Set the front panel controls with the following values:
 - **DAQmx Channel Name:** Sine Wave
 - **Rate:** 10000
 - **Samples per Channel:** 3000
 3. Connect the sine wave output to analog input ch1 on the DAQ Signal Accessory.

Block Diagram

4. Build the following block diagram.

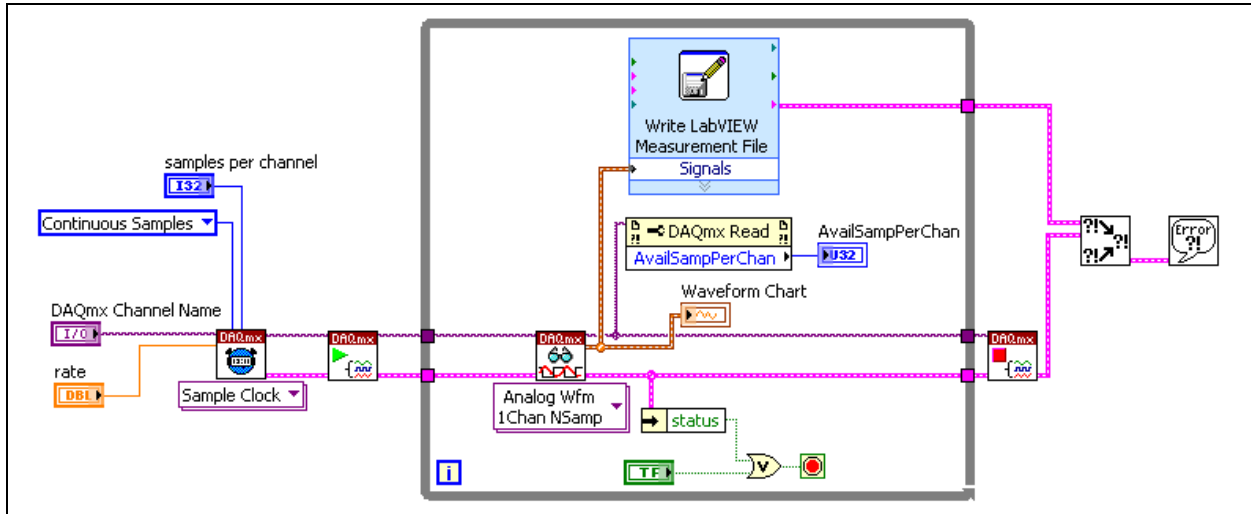


- Place the DAQmx Read Property Node, located on the **Functions»All Functions»NI Measurements»DAQmx - Data Acquisition** palette, on the block diagram. Use the Property Node to obtain additional information about the buffer status. Right-click the node and select **Properties»Status»Available Samples per Channel** from the shortcut menu. This property monitors the backlog amount of samples per channel in the buffer.
 - Place the Unbundle by Name function, located on the **Functions»All Functions»Cluster** palette, on the block diagram. This function accepts the Boolean **status** value from the error cluster.
 - Place the Simple Error Handler VI, located on the **Functions»All Functions»Time & Dialog** palette, on the block diagram. In the event of an error, this VI displays a dialog box with information regarding the error and its location.
- Save the VI as `Continuous Acquire.vi` in the `C:\Exercises\LabVIEW DAQ` directory.
 - Display the front panel. Run the VI and monitor the data plotted on the graph as you change the frequency knob on the DAQ Signal Accessory. Data fills a buffer of fixed size in memory then overwrites values from the beginning of the buffer.
 - Monitor **Available Samples per Channel** as you decrease the **rate** or the **samples per channel**. The available samples per channel is defined as the number of samples per channel acquired into the acquisition buffer but not read. It is a measure of how well you are keeping up with a continuous acquisition. If the backlog steadily increases, you are not reading data fast enough from the buffer and will eventually lose data. If this occurs, the DAQmx Read VI returns an error.

Streaming to Disk

Modify the VI to stream the acquired data to disk.

8. Select **File»Save As** and save the VI as Continuous Acquire with File IO.vi in the C:\Exercises\LabVIEW DAQ directory.
9. Modify the block diagram as shown in the following figure.

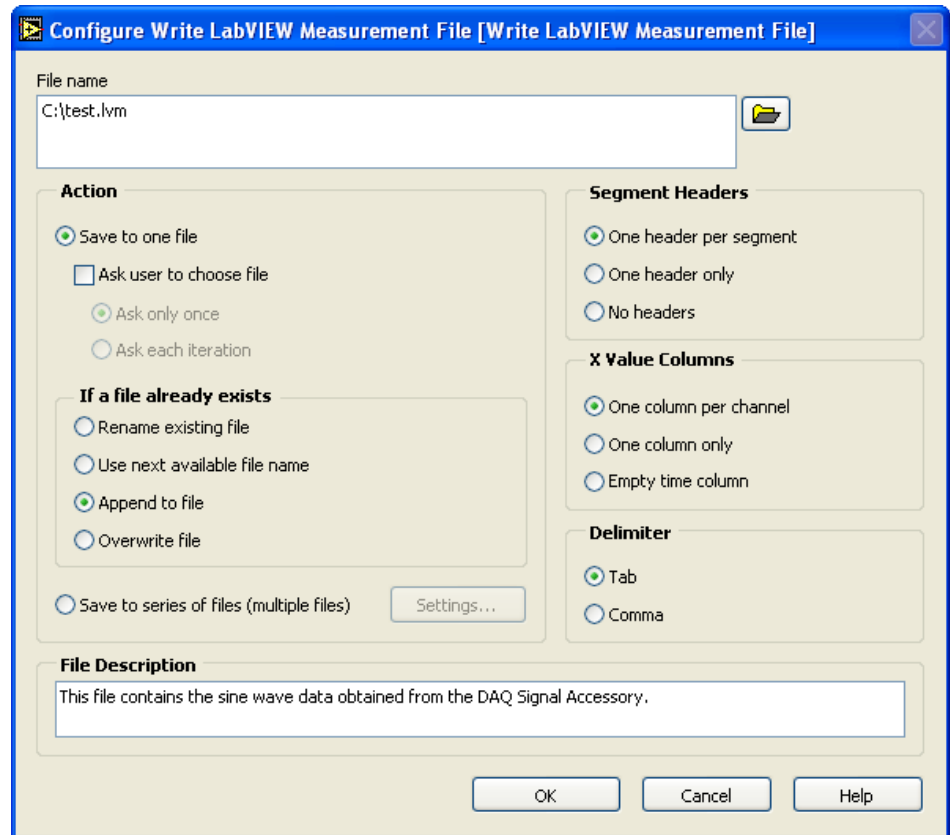


- a. Place the Merge Errors VI, located on the **Functions»All Functions»Time & Dialog** palette, on the block diagram. This VI merges error clusters from different functions into a single error cluster.

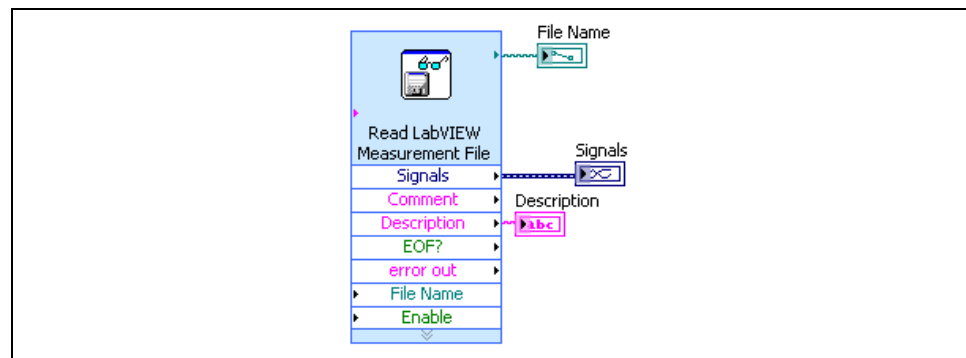


- b. Place the Write LabVIEW Measurement File Express VI, located on the **Functions»Output** palette, on the block diagram. This Express VI writes LabVIEW measurement data to a file.

- In the **Configure Write LabVIEW Measurement File** dialog box that appears, select the following settings:

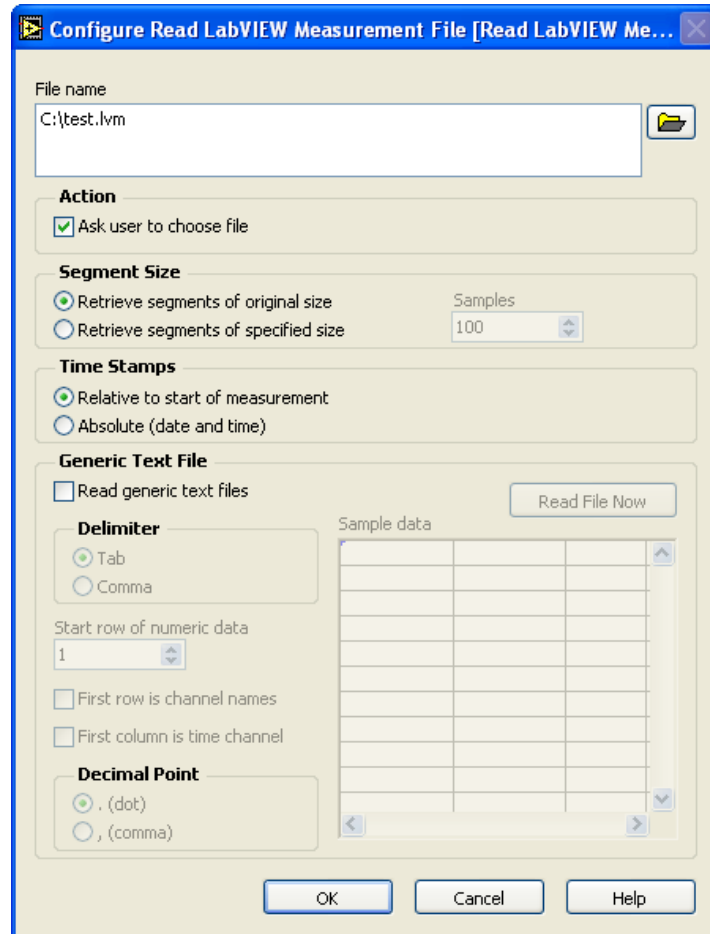


10. Click the **OK** button to exit the **Configure Write LabVIEW Measurements File** dialog box.
11. Save and run the VI.
12. Open a blank VI and create the following block diagram to build a VI that reads the LabVIEW Measurement File.





- a. Place the Read LabVIEW Measurement File VI, located on the **Functions»Input** palette, on the block diagram. This Express VI reads LabVIEW measurement data from a file.
 - In the **Configure Read LabVIEW Measurement File** dialog box that appears, select the following settings:



13. Click the **OK** button to apply the changes and close the **Configure Read LabVIEW Measurements File** dialog box.
14. On the block diagram, resize the Read LabVIEW Measurement File Express VI to display more output terminals.
15. Right-click the **File Name** and **Description** outputs and select **Create»Indicator** from the shortcut menu. Right-click the **Signals** output and select **Create»Graph Indicator** from the shortcut menu.
16. Run the VI. The signal that you acquired in the previous VI now appears in the waveform graph.

17. Save the VI as Read Data File.vi in the
C:\Exercises\LabVIEW DAQ directory.
18. Close all VIs.

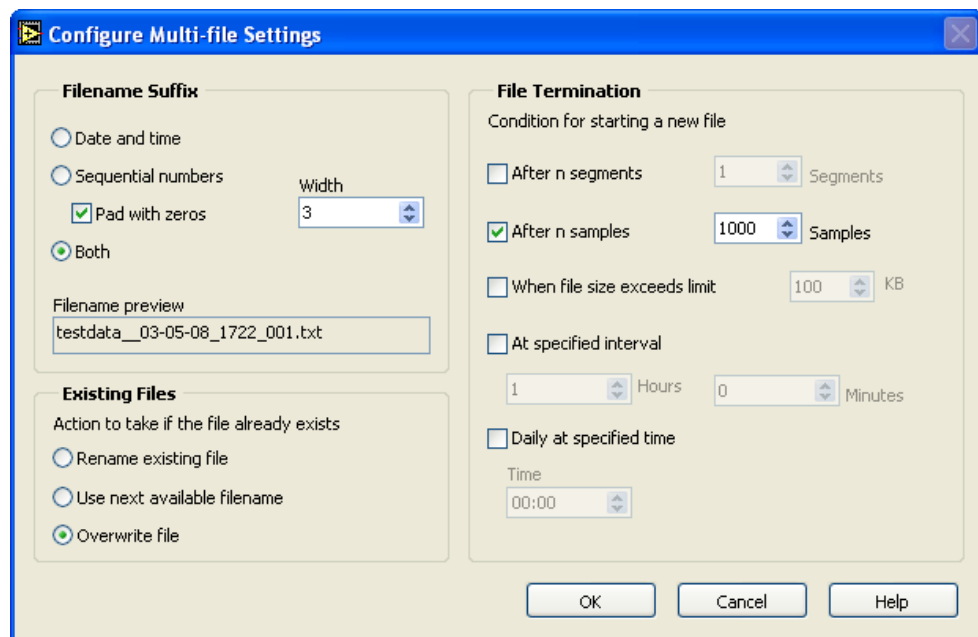
End of Exercise 4-6

Exercise 4-7 Streaming Data to Multiple Files

Objective: To use the Write LabVIEW Measurement File Express VI to stream data to multiple files.

For large amounts of data, it might be more convenient to write the data to multiple files so you can organize and manage the data files. You can configure the Write LabVIEW Measurement File Express VI to write to multiple files. Writing to multiple files also can prevent data loss in the case of a program crash or failure.

1. Open the Continuous Acquire with File IO.vi located in the C:\Exercises\LabVIEW DAQ directory.
2. Double-click the Write LabVIEW Measurement File Express VI to open the **Configure Write LabVIEW Measurement File** dialog box.
 - a. In the **Action** section of the dialog box, select the **Save to series of files (multiple files)** option.
 - b. Click the **Settings** button to open the **Configure Multi-file Settings** dialog box.
 - c. Configure the **Configure Multi-file Settings** dialog box as shown in the following figure.



- d. Click the **OK** button to exit the **Configure Multi-file settings** dialog box.
- e. Click the **OK** button to exit the **Configure Write LabVIEW Measurement File** dialog box.

3. Run the VI for several seconds.
4. Open the Read Data File VI located in the C:\Exercises\LabVIEW DAQ directory and switch to the block diagram.
5. Double-click the Read LabVIEW Measurement File Express VI to open the **Configure Read LabVIEW Measurement File** dialog box.
 - a. In the **Action** section, place a checkmark in the **Ask User to choose file** checkbox.
 - b. Click the **OK** button to close the dialog box.
6. Run the VI. You should be prompted to select a file to read. Navigate to the C:\ directory and notice the data files that were created.
7. Select one of the files to read and click the **OK** button.
8. Run the VI again and select a different file to read.
9. Save and close all VIs.

End of Exercise 4-7

Exercise 4-8 Trigger Analog Input on Digital Edge

Objective: To use a digital trigger to start a continuous data acquisition operation.

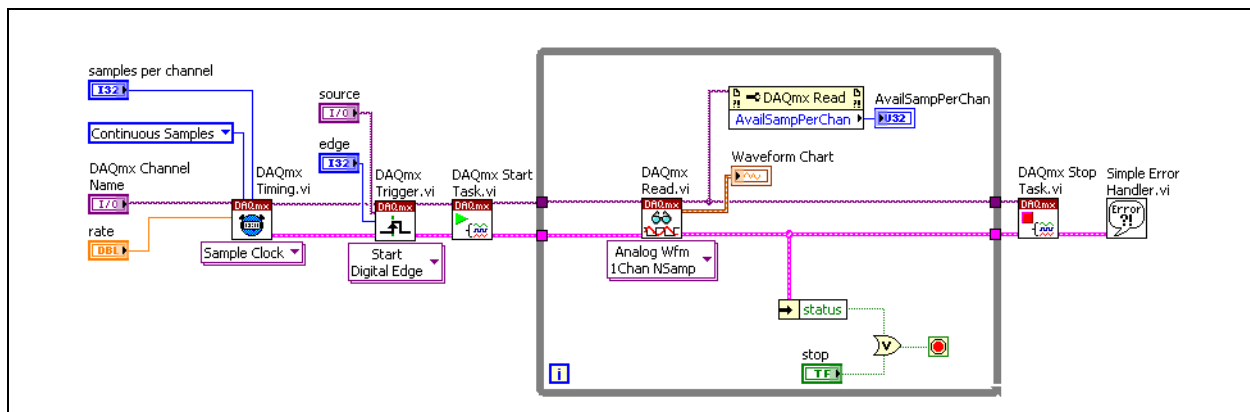
Complete the following steps to build a VI that uses a digital trigger to start a continuous data acquisition operation.

1. Open the Continuous Acquire VI located in the C:\Exercises\LabVIEW DAQ directory.
2. Select **File»Save As** and save the VI as Trigger Continuous Acquire.vi in the C:\Exercises\LabVIEW DAQ directory.
3. Connect the sine wave output to analog input CH1 on the DAQ Signal Accessory.

Block Diagram



4. Open the block diagram and place the DAQmx Trigger VI, located on the **Functions»All Functions»NI Measurements»DAQmx - Data Acquisition** palette, on the block diagram. This VI configures the trigger settings. Right-click the **edge** and **source** inputs and select **Create»Control** from the shortcut menu as shown in the following figure.



5. Open the front panel and enter the following settings:
 - **DAQmx Channel Name:** Sine Wave
 - **Samples per channel:** 1000
 - **Rate:** 5000
 - **Source:** /DevX/PFI0, where X corresponds to the device number of your DAQ device, as set in MAX.
 - **Edge:** Rising

6. Save the VI.
7. Run the VI. The VI does not acquire data until the trigger occurs. Press the digital trigger button on the DAQ Signal Accessory to begin continuously acquiring data.
8. Stop and close the VI when you finish.

End of Exercise 4-8

Exercise 4-9 NI-DAQmx Code Generation

Objective: To use the NI-DAQmx code generation features to generate code for an analog input operation.

The code generation feature of NI-DAQmx enables you to perform rapid prototyping and development. For more advanced applications, implement the programming techniques learned in this course for data acquisition programs.

Front Panel

1. Place a DAQmx Task Name Control, located on the **Controls»All Controls»I/O»DAQmx Name Controls** palette, on the front panel.
2. Select **My Voltage Task** from the pull-down menu.
3. Use the DAQ Assistant to configure the settings, timing, and triggering for the task. In the **Task Timing** tab, select **Acquire Continuously**. Leave all other settings at their default values.
4. Click the **OK** button to exit the DAQ Assistant.
5. Right-click the task and select **Generate Code»Configuration and Example** from the shortcut menu.

You can select from the following options for code generation:

- **Example**—Generates example code to perform the measurement you configured in the DAQ Assistant.
 - **Configuration**—Generates code to programmatically configure the task using the DAQmx Create Virtual Channel VI.
 - **Configuration and Example**—Generates code to programmatically configure the task and demonstrate an example measurement or generation.
6. The block diagram automatically appears with the generated code. Notice that the DAQmx Task Name Control was converted into a subVI.
 7. Double-click the subVI and view the block diagram. Open the **Context Help** window by pressing the <Ctrl-H> keys or selecting **Help»Show Context Help**. Hover the mouse over each function to display more information about the function in the **Context Help** window.
 8. Close all VIs. Do not save changes.

End of Exercise 4-9

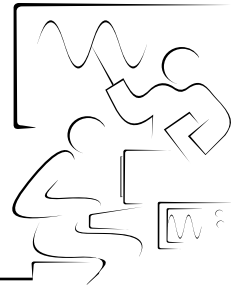
Summary

- Use the Nyquist theorem to determine how fast you need to sample.
- DAQ Channel Names and the waveform data type make DAQ programming easy and flexible.
- Single-point acquisition is software timed, non-buffered, and useful for slow changing signals.
- Round-robin, interval, and simultaneous sampling affect phase relationships of signals differently.
- Buffered acquisition is hardware timed with a regular buffer.
- Continuous acquisition is hardware timed with a circular buffer.
- Buffered or continuous acquisition can be triggered with a digital or analog signal. If the DAQ device does not support analog triggering, use conditional retrieval.
- You can stream data to disk for later analysis and presentation.
- Use the NI-DAQmx code generation feature for rapid prototyping.

Notes

Lesson 5

Signal Conditioning



This lesson introduces you to using signal conditioning with a data acquisition system to accurately measure a wide variety of physical phenomena. This lesson also describes signal conditioning methods to improve signal quality.

You Will Learn:

- A. Overview of signal conditioning
- B. Signal conditioning configuration
- C. Signal conditioning functions
- D. Transducer conditioning

A. Overview of Signal Conditioning

A typical data acquisition system consists of a physical phenomenon, transducers, signal conditioning, data acquisition device, and a computer.

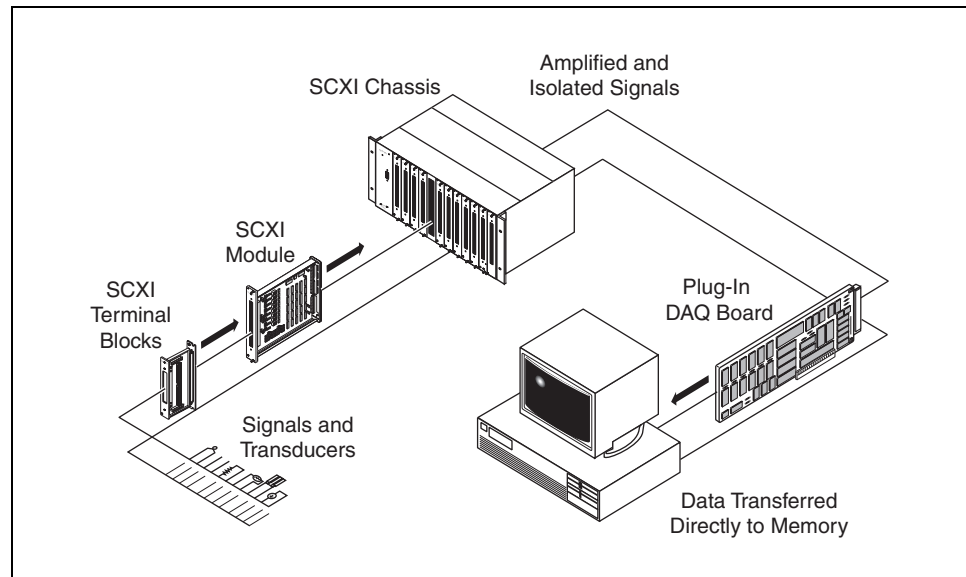
The signal source is typically a transducer to measure a physical phenomenon. Different transducer manufacturers and transducer types measure a wide variety of signal levels. The signal is typically connected to a signal conditioning device. This signal connection is usually two or more wires varying in length from inches to miles, perhaps traversing through high voltage or other electromagnetic hazards. Most signal quality problems originate in the signal connection section.

Most real-world sensors and transducers generate signals that you must condition before a DAQ device can reliably and accurately acquire the signal. This front-end processing, referred to as signal conditioning, includes functions such as signal amplification, filtering, electrical isolation, and multiplexing. Signal conditioning systems can be found on the data acquisition device or in separate devices such as Signal Conditioning eXtensions for Instrumentation (SCXI) or Signal Conditioning Components (SCC). The goal is to clean up the signal before it is measured by the analog-to-digital converter (ADC).

The data acquisition device is typically a computer plug-in device such as a National Instruments DAQ device. The data acquisition device provides the capability to digitize the conditioned analog signal. The computer can then analyze and present the digitized and conditioned signal.

B. Signal Conditioning Configuration

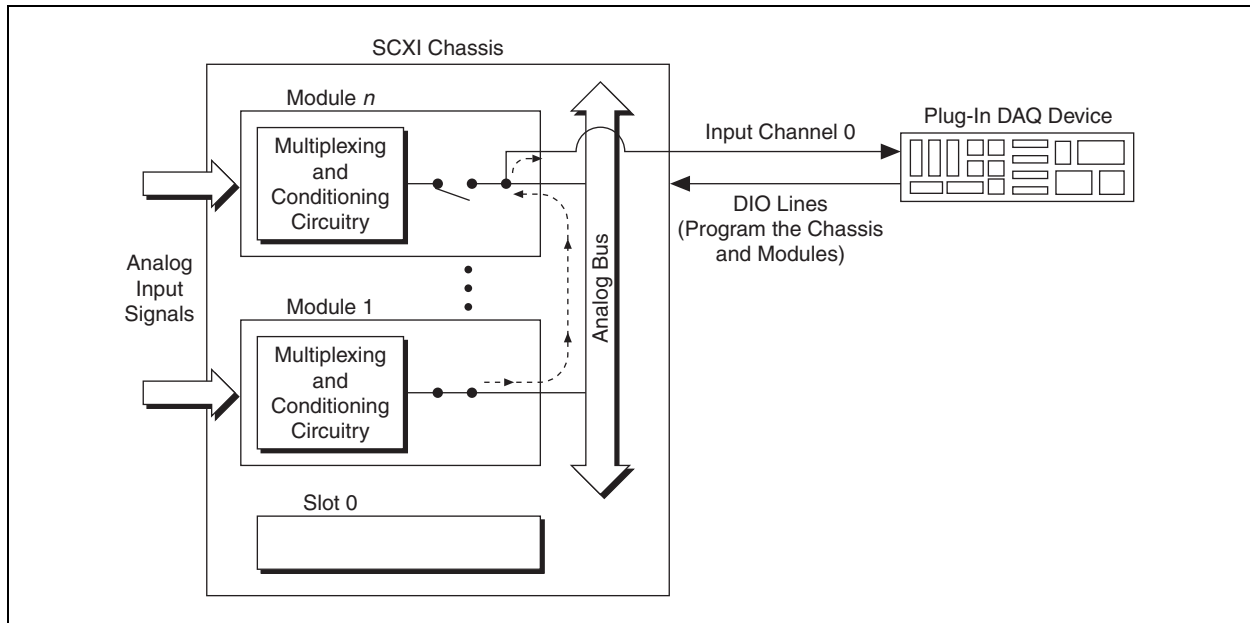
SCXI is a complete signal conditioning architecture that provides a versatile, high-performance signal conditioning platform. The following illustration shows the basic system components of a signal conditioning system.



Transducers are connected to the terminal block. The SCXI chassis houses the SCXI modules, supplying power and controlling the SCXibus. The SCXI chassis is connected to a plug-in DAQ device within the computer. The DAQ device controls the operation of the SCXI chassis.

The following illustration shows the architecture of the SCXI signal conditioning system.

During single-channel readings, the DAQ device serially writes a digital pattern to SCXI Slot 0, located in the chassis, indicating which SCXI module to address. The DAQ device then writes a digital pattern to the module indicating which input channel is read, which configures the module to route the desired signal to the analog bus of the SCXI chassis. Ultimately, the signal is routed to the analog input channel of the DAQ device. The DAQ device then reads channel 0, as shown in the following illustration. The NI-DAQ device driver performs all of this low-level digital communication and signal routing when you call the single-channel analog input functions.



With multichannel scanning, the DAQ device programs SCXI Slot 0 with a list of modules and the number of channels to scan from each module. Each module in the list is programmed with the channel on which to start the scan. The DAQ device or module then begins the multichannel scan. The SCANCLK signal from the DAQ device synchronizes the SCXI multiplexing with the internal clock that triggers the A/D conversions on the DAQ device. SCXI Slot 0 enables and disables the modules according to the preprogrammed list. In this way, the system multiplexes channels from several modules to an analog input channel of the DAQ device at very high rates. NI-DAQ contains the high-level functions that perform multichannel scanning operations.

Complete the following steps to set up the SCXI system.

1. Make sure that the correct cable assembly and cable adapter are connected to the correct module.
2. Connect the module to the DAQ device.

3. Make sure that the correct terminal blocks are connected to the correct module.
4. Power on the SCXI chassis.



Caution SCXI modules are *not* hot-swappable. Make sure the chassis is powered off before removing or adding any modules, terminal blocks, or cables.

Exercise 5-1 SCXI Configuration using Measurement & Automation Explorer

Objective: To use MAX to configure a signal conditioning system for a PC-based DAQ system.



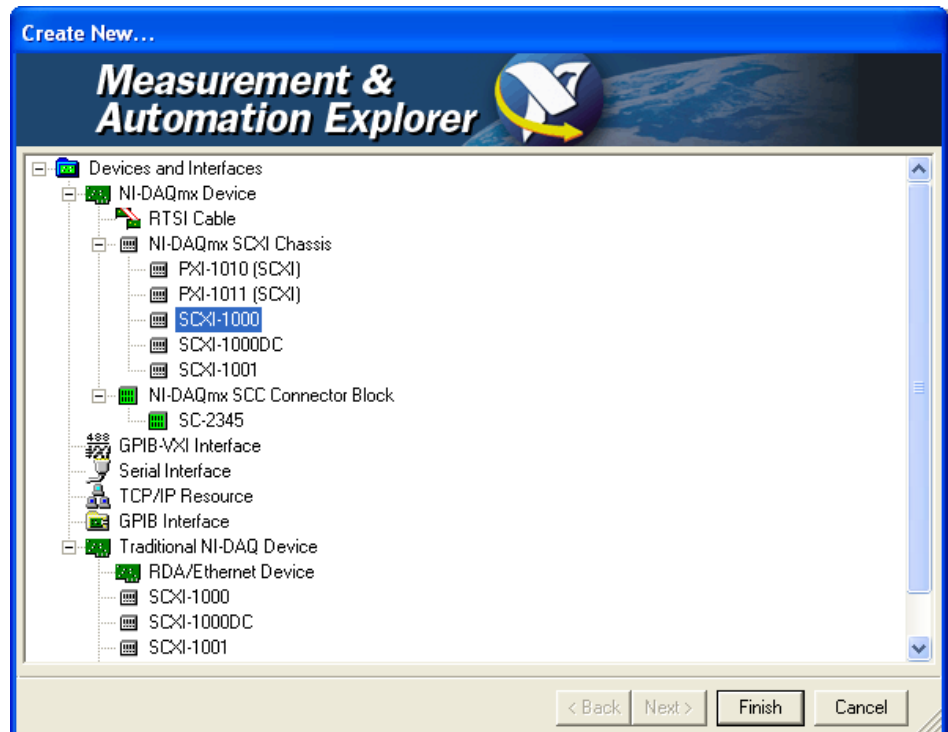
Caution SCXI modules are *not* hot-swappable. Make sure the chassis is powered off before removing or adding any modules, terminal blocks, or cables.

1. Power off the SCXI chassis.
2. Review the SCXI-1000 4-slot chassis, modules, and terminal blocks. The chassis should contain the following modules in order from Slot 1 (leftmost slot) to Slot 4 (rightmost slot). The following list includes modules you will use in later exercises.
 - Slot 1—SCXI-1520 8 channel universal strain gauge input module
 - SCXI-1314 universal strain gauge terminal block
 - Slot 2—SCXI-1180 feedthrough panel

The SCXI-1180 allows you to access the unused channels on the DAQ device from the front of the SCXI chassis.

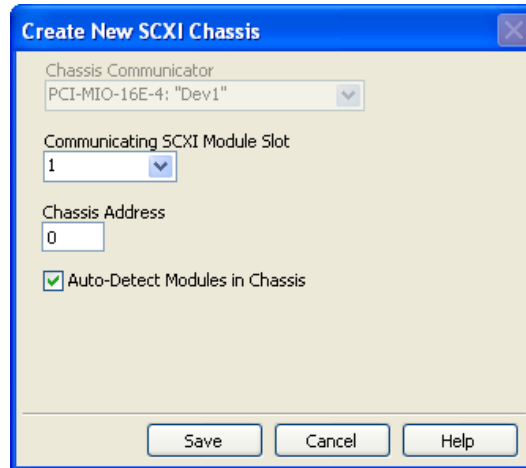
 - SCXI-1302 general purpose terminal block
 - Slot 3—SCXI-1125 8 channel isolation amplifier module
 - SCXI-1327 attenuation terminal block
 - Slot 4—SCXI-1141 8 channel elliptic lowpass filter module
 - SCXI-1304 signal ground referencing terminal block
3. Look at the back of the SCXI-1000 chassis. The module in slot 1 should have an SCXI-1349 cable adapter for E Series DAQ devices plugged into it. Notice how the SCXI-1349 has a 68-pin cable connector on its back and a 50-pin connector on its side. The 68-pin back connector is for connecting to the cable coming from the DAQ device in the computer. The 50-pin side connector is for accessing the unused channels on the DAQ device.
4. The SCXI-1180 (slot 2) is not a module but a feedthrough panel that contains a cable that connects to the side connector of the SCXI-1349 cable adapter and allows you to access the unused channels on the DAQ device from the front of the SCXI chassis.
5. Connect the DAQ device to the SCXI chassis and gain access to the unused channels of the DAQ device.
 - a. Take the cable coming from the DAQ device in the computer and detach it from the DAQ Signal Accessory.
 - b. Make sure the SCXI-1349 cable adapter is connected to the back of the SCXI-1520 in slot 1 of the SCXI chassis.

- c. Plug the DAQ cable into the 68-pin back connector of the SCXI-1349.
6. Power on the SCXI chassis.
7. Launch MAX by double-clicking the icon on the Windows desktop. Right-click **Devices and Interfaces** and select **Create New** from the shortcut menu.
8. Select the **Devices and Interfaces»NI-DAQmx Device»NI-DAQmx SCXI Chassis»SCXI-1000** device and click the **Finish** button.

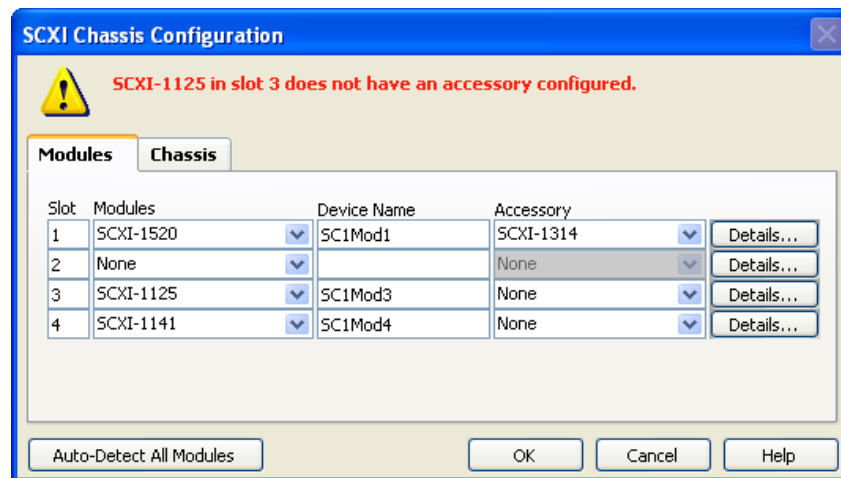


9. Select the SCXI module slot that will communicate with the DAQ device. In this case, the first module, the SCXI-1520, has the SCXI-1349 cable adapter connected on the backplane and will communicate with the DAQ device. Select **1** from the **Communicating SCXI Module Slot** pull-down menu.

10. In the Create New SCXI Chassis dialog box, set **Chassis Address** to **0**. Place a checkmark in the **Auto-Detect Modules in Chassis** checkbox and click the **Save** button.



11. The **SCXI Chassis Configuration** dialog box should appear as shown in the following figure.



Note The 1180 is a feedthrough panel and will not be detected as a module. Its slot appears to be empty in the **SCXI Chassis Configuration** dialog box.

- For the SCXI-1125 module, select **SCXI-1327** from the **Accessory** pull-down menu.
- For the SCXI-1141 module, select the **SCXI-1304** from the **Accessory** pull-down menu.
- Click the **OK** button to close the dialog box.

12. Look under **Devices and Interfaces** and then **NI-DAQmx Devices** to expand the tree to show **SCXI-1000: "SC1"**. Next, double-click **SCXI-1000: "SC1"** to expand the tree to show the modules installed.
13. Right-click **SCXI-1000: "SC1"** and select **Test** from the shortcut menu. Click the **OK** button. If the chassis does not pass the verification test, inform the instructor.
14. Click each of the modules and observe the **Attributes** tab that displays information about the module properties you set in the **SCXI Chassis Configuration** dialog box.
15. Exit MAX.

End of Exercise 5-1

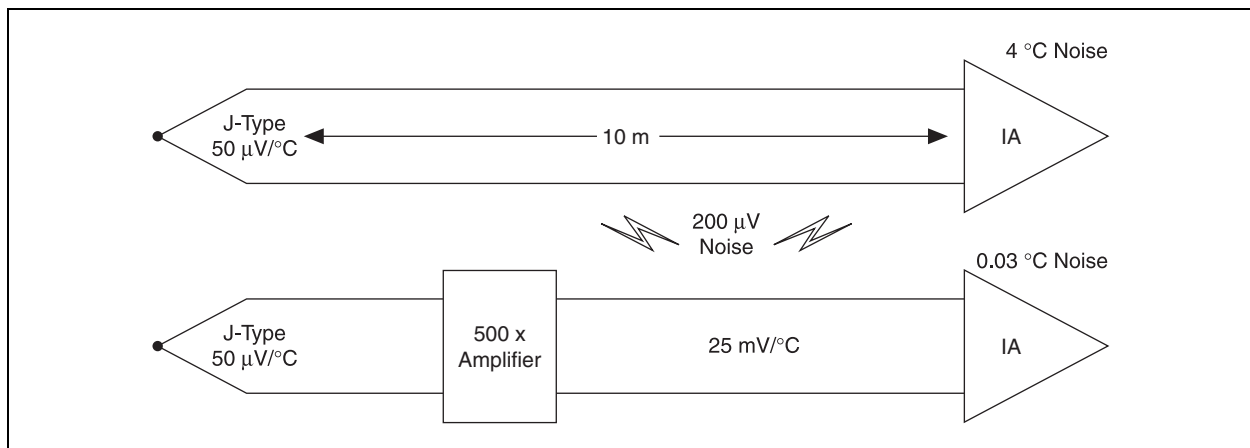
C. Signal Conditioning Functions

In addition to handling specific transducers, signal conditioning devices perform a variety of general-purpose conditioning functions to improve the quality, flexibility, and reliability of a measurement system.

Signal Amplification

Because real-world signals are often very small in magnitude, signal conditioning can improve the accuracy of data. Amplifiers boost the level of the input signal to better match the range of the ADC, increasing the resolution and sensitivity of the measurement. Though many DAQ devices include onboard amplifiers, many transducers, such as thermocouples, require additional amplification.

Many transducers produce voltage output signals of millivolts or even microvolts. Amplifying these low-level signals directly on a DAQ device also amplifies any noise picked up from the signal connections. When the signal is small, even a small amount of noise can drown out the signal itself, leading to erroneous data. A simple method for reducing the signal-to-noise ratio is to amplify the signal as close to the source as possible. This boosts the signal above the noise level before noise in the connections can corrupt the signal and improves the signal-to-noise ratio of the measurement. For example, the following illustration shows a J-type thermocouple outputting a low-level voltage signal that varies by about $50 \mu\text{V}/^\circ\text{C}$.



Suppose the thermocouple leads travel 10 m through an electrically noisy environment to the DAQ system. If the noise sources in the environment couple $200 \mu\text{V}$ onto the thermocouple leads, you get a temperature reading with about 4°C of noise. Amplify the signal close to the thermocouple, before noise corrupts the signal to reduce the effect on the final measurement. Amplifying the signal with a gain of 500 near the thermocouple produces a thermocouple signal that varies by about

25 mV/°C. As this high-level signal travels the same 10 m, the 200 μV of noise coupled onto this signal after the amplification has less effect on the final measurement, adding only 0.03 °C of noise.

Exercise 5-2 Setting Gain

Objective: To set the gain on channel 0 and channel 1 of the SCXI-1125 and observe the differences of the gain settings on a chart in LabVIEW.



Caution SCXI modules are *not* hot-swappable. Make sure the chassis is powered off before removing or adding modules, terminal blocks, or cables.

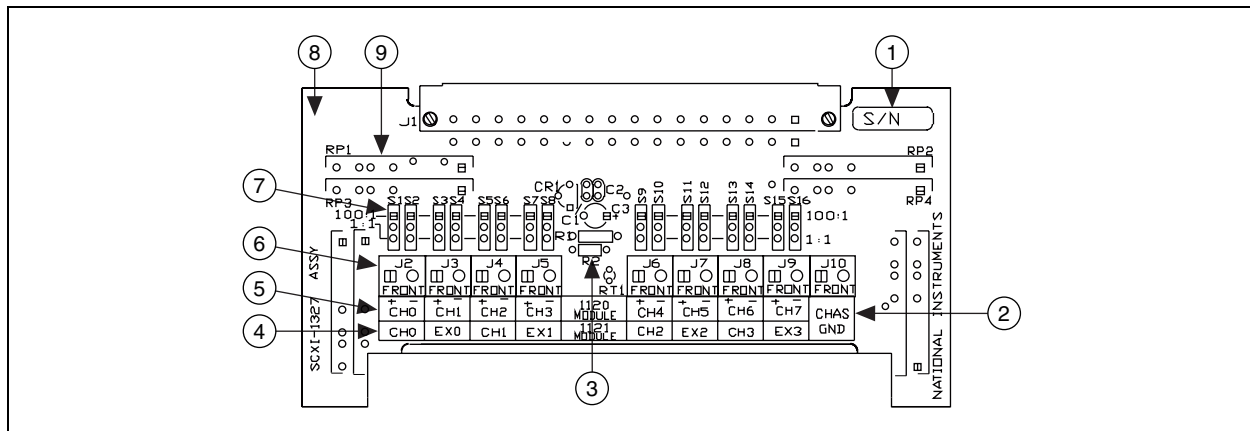
1. Launch MAX and expand **Devices and Interfaces»NI-DAQmx Devices»SCXI-1000: “SC1”**.
2. Right-click **3: SCXI-1125: “SC1Mod3”** and select **Properties** from the shortcut menu. Click the **Configure** button. Click the **Gain Configuration** tab and configure channel 0 and channel 1 with the following settings:
 - **Channel: 0 Attenuation: 1.00**
 - **Channel: 1 Attenuation: 1.00**
3. Click **OK** twice to exit the **Details: SCXI-1125 SC1Mod3** dialog box.

Connect the Thermocouple to Channels 0 and 1 of the SCXI-1327 Terminal Block



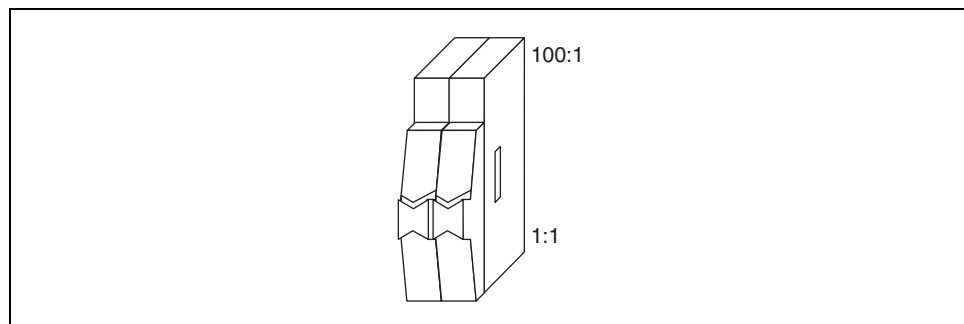
Caution Before proceeding, make sure the SCXI-1327 terminal block is *not* attached to the SCXI-1125. If it is, power off the chassis first and remove the SCXI-1327 by loosening the top and bottom screws that attach it to the SCXI-1125 module.

- Remove the top cover of the SCXI-1327 attenuation terminal block. Examine the following illustration and match the numbered components with the components on the SCX-1327 terminal block.



- | | |
|--|--|
| 1 Serial Number | 6 Screw Terminals |
| 2 Chassis Ground | 7 Switches to Enable or Bypass the Attenuator |
| 3 Thermistor | 8 Product Name, Assembly Number, and Revision Letter |
| 4 Channel Labeling for the SCXI-1121 | 9 Voltage Dividers |
| 5 Channel Labeling for the SCXI-1120, SCXI-1120D, SCXI-1125, SCXI-1126 | |

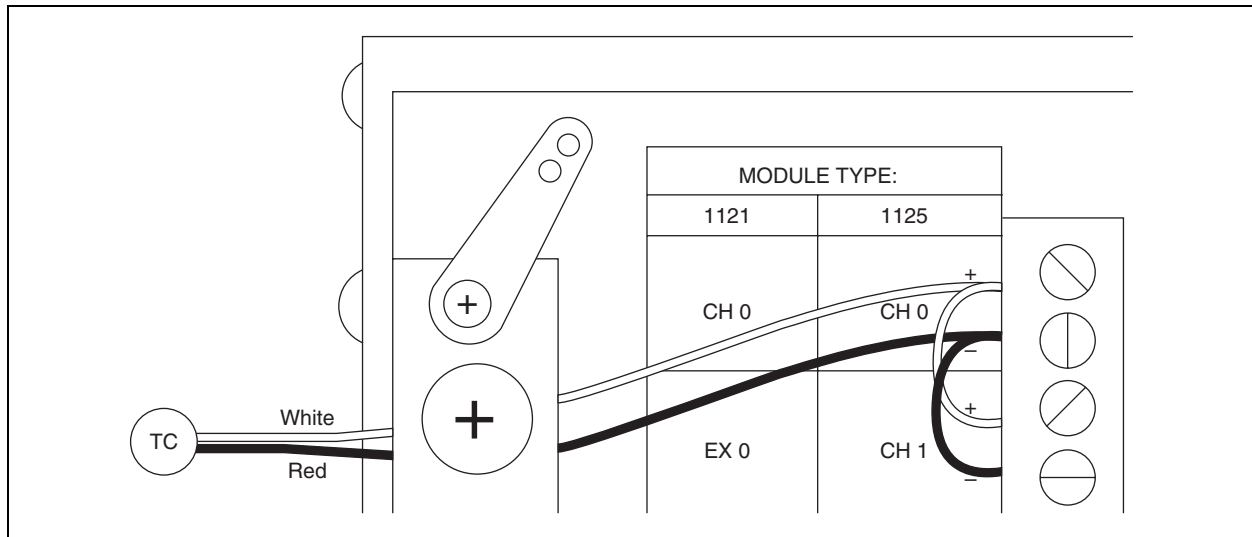
- Locate the switches used to bypass the attenuation circuitry on CH 0 (S1 and S2) and CH 1 (S3 and S4). Make sure they are all set to 1:1 as shown in the following illustration.



6. Using the thermocouple, make the connections shown in the following illustration.



Note Use wires that are the same type of metal as the thermocouple. The white wire is iron, the red wire is constantan. Check with your instructor to make sure you use the correct type of wires.



- Attach the white wire from the thermocouple to CH 0+ and the other wire from the thermocouple to CH 0-.
- Take an additional piece of each type of wire you used to build the thermocouple and attach them to CH 0 as follows.
 - Attach the white wire to the positive (+) screw terminal of CH 0 and tighten the screw in place. Make sure the wire and the thermocouple are both held in place.
 - Attach the red wire to the negative (–) screw terminal of CH 0 and tighten the screw in place. Make sure the wire and the thermocouple are both held in place.
- Attach the white wire to CH 1+.
- Attach the red wire to CH 1-.
- Replace the SCXI-1327 cover and tighten the cover screws.
- Attach the SCXI-1327 to the front of the SCXI-1125 and tighten the top and bottom screws to secure the terminal block to the module.
- Power on the chassis.

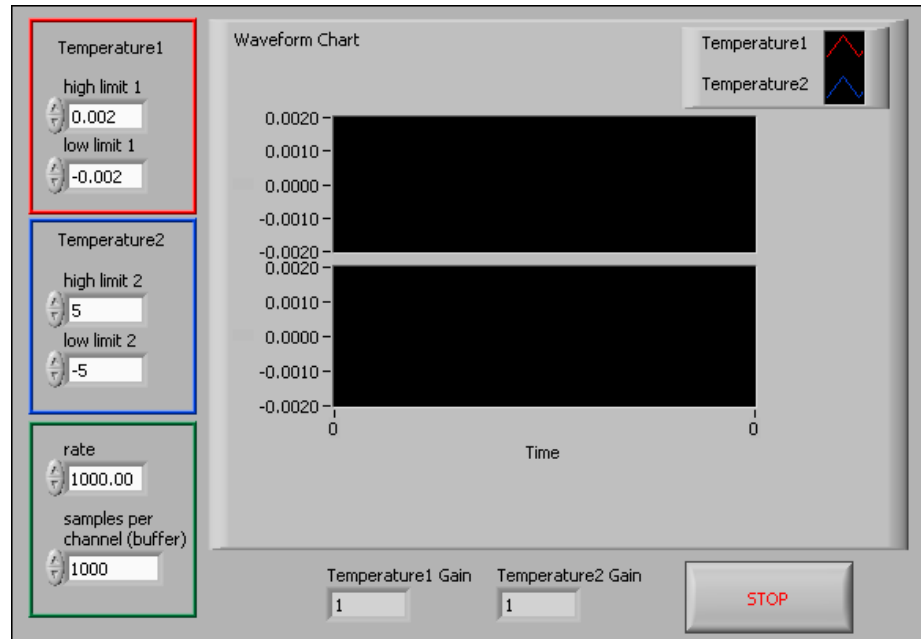
7. Return to MAX and right-click **NI-DAQmx Global Channels** under **Data Neighborhood**. Select **Create New NI-DAQmx Channel** from the shortcut menu. Configure the following settings for the first channel:
 - **Measurement Type:** Analog Input
 - **Sensor Type:** Voltage. For this exercise, the thermocouple is treated as a very small voltage source instead of as a temperature source. You create a thermocouple channel later in the course.
 - **Physical Channel:** SC1Mod3/ai0
 - **Name:** Temperature1
 Configure the following settings for the second channel:
 - **Measurement Type:** Analog Input
 - **Sensor Type:** Voltage
 - **Physical Channel:** SC1Mod3/ai1
 - **Name:** Temperature2
8. Right-click **Data Neighborhood** and select **Create New»NI-DAQmx Task** from the shortcut menu. Configure the following settings:
 - **Measurement Type:** Analog Input
 - **Sensor Type:** Voltage
 - **Channels:** Select the **Add Existing DAQmx Global Channels** option and select `Temperature1` and `Temperature2`.
 - **Name:** `SmallVoltageTask`
9. In the **Analog Input Voltage Task Configuration** dialog box, configure the task to continuously acquire data.
10. Click the **Start** button.
11. Use your finger or a glass of ice water to test the thermocouple and determine if the reading changes accordingly. If no change is detected or you get an error, inform the instructor.
12. Click the **Stop** button.
13. Click the **OK** button to exit the test panel.
14. Click the **Save Task** button.
15. Exit MAX. Save any unsaved NI-DAQmx channels.

Setting Gain in LabVIEW

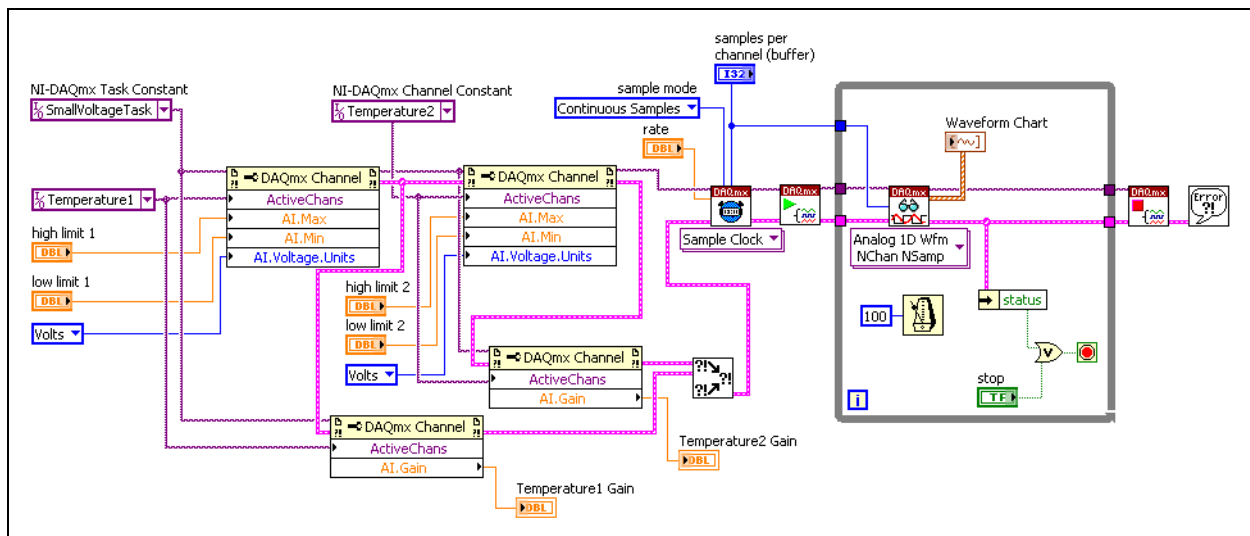
Complete a VI that acquires data from the thermocouple you connected to the SCXI-1125/SCXI-1327. This VI demonstrates how to programmatically set the gain of individual channels by adjusting the signal range (high limits and low limits) of the signal in LabVIEW and shows, by comparison, the

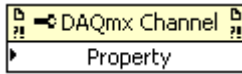
benefits of using gain when acquiring small voltages, such as the millivoltage output of a thermocouple.

1. Launch LabVIEW and open the SCXI-1125 Gain vs No Gain VI located in the C:\Exercises\LabVIEW DAQ directory. The following front panel displays.



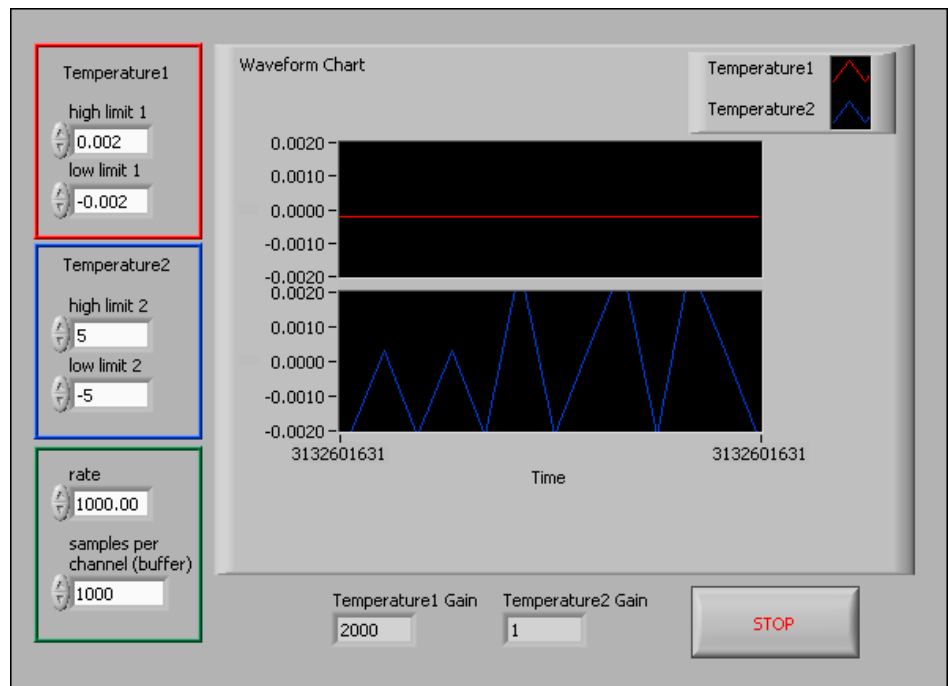
2. Modify the block diagram as shown in the following figure.





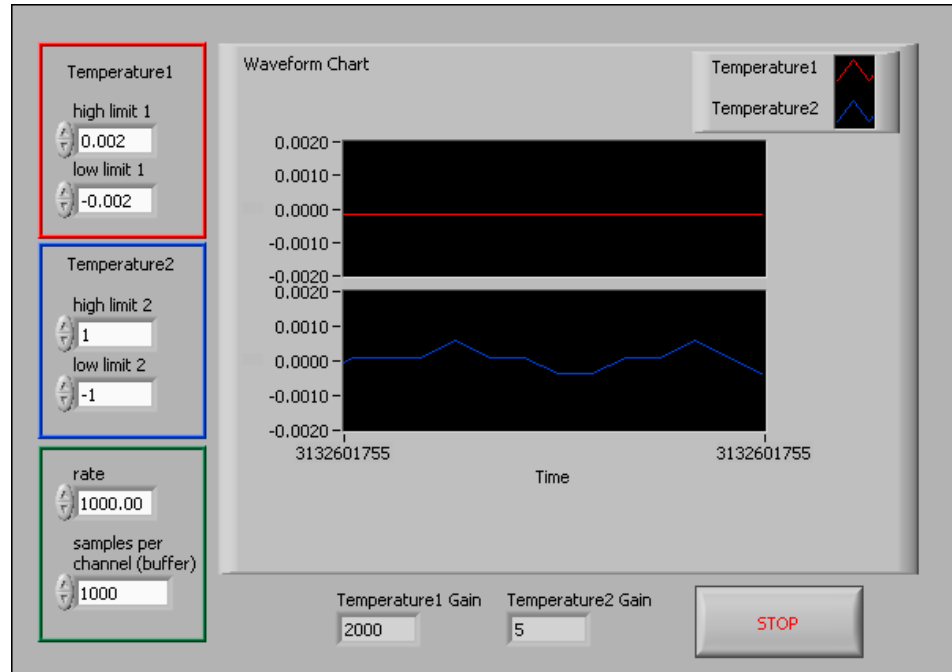
- a. Place the DAQmx Channel Property Node, located on the **Functions»All Functions»NI Measurements»DAQmx - Data Acquisition** palette, on the block diagram. This Property Node reads and/or writes properties pertaining to the specific DAQmx Channel specified by the ActiveChans property. Use four of these property nodes to set input limits and read the gain settings.
- b. Place the DAQmx Timing VI, located on the **Functions»All Functions»NI Measurements»DAQmx - Data Acquisition** palette, on the block diagram. This VI configures the sample timing rate and sampling mode. Right-click the **sample mode** input and select **Create»Constant** from the shortcut menu.
- c. Place the Merge Errors VI, located on the **Functions»All Functions»Time & Dialog** palette, on the block diagram. This VI merges multiple error clusters into one error.
- d. Place the DAQmx Start Task VI, located on the **Functions»All Functions»NI Measurements»DAQmx - Data Acquisition** palette, on the block diagram. This VI starts a task operation.
- e. Place the DAQmx Read VI, located on the **Functions»All Functions»NI Measurements»DAQmx - Data Acquisition** palette, on the block diagram. This polymorphic VI reads sample data. Select the **Analog»Multiple Channels»Multiple Samples»1D Waveform** instance from the pull-down menu.
- f. Place the DAQmx Stop Task VI, located on the **Functions»All Functions»NI Measurements»DAQmx - Data Acquisition** palette, on the block diagram. This VI stops a task operation.
- g. Place the Simple Error Handler VI, located on the **Functions»All Functions»Time & Dialog** palette, on the block diagram. In the event of an error, this VI displays a dialog box with information about the error and where it occurred.
- h. Place the Wait until Next ms Multiple function, located on the **Functions»All Functions»Time & Dialog** palette, on the block diagram. This function waits until the value of the millisecond timer becomes a multiple of the specified millisecond multiple. This function controls the While Loop execution rate. Set the millisecond multiple to 100.
- i. Place the Or function, located on the **Functions»Arithmetic & Comparison»Express Boolean** palette, on the block diagram.
- j. Place the Unbundle by Name function, located on the **Functions»All Functions»Cluster** palette, on the block diagram. This function returns the error **status** Boolean value. If an error occurs, execution of the While Loop stops.

3. Save the VI.
4. On the front panel, set the following controls:
 - **rate:** 1000
 - **samples per channel (buffer):** 1000
 - **High and Low Limits:**
 - **Temperature1:**
 - **High Limit:** 0.00200
 - **Low Limit:** -0.00200
 - **Temperature2:**
 - **High Limit:** 5.00000
 - **Low Limit:** -5.00000
5. Run the VI.
6. Notice the gain settings for the two channels. Place your finger on the thermocouple and notice the difference in the two channels on the chart, as shown in the following figure.



7. Stop the VI. Change the **High Limit** of **Temperature2** to 1.00000 and the **Low Limit** of **Temperature2** to -1.00000. Run the VI again.

8. Place your fingers on the thermocouple again and pay close attention to the signal of CH 1. Notice the signal changes from a low voltage to a high voltage, but you are unable to accurately read the voltages between the two levels, as shown in the following figure.



You are unable to accurately read the voltages between the two levels due to code width, the smallest change in the signal the system can detect. A range of 2 V (–1.00 V to 1.00 V) divided by the product of the gain (5.00) and resolution (212) results in a code width of 0.097 mV.

9. Notice the gain setting for Temperature2 has changed. Experiment with the VI by stopping it and changing the **High Limits** and **Low Limits** values and notice how different values affect the gain and the signals.
10. Save and close the VI when you finish.

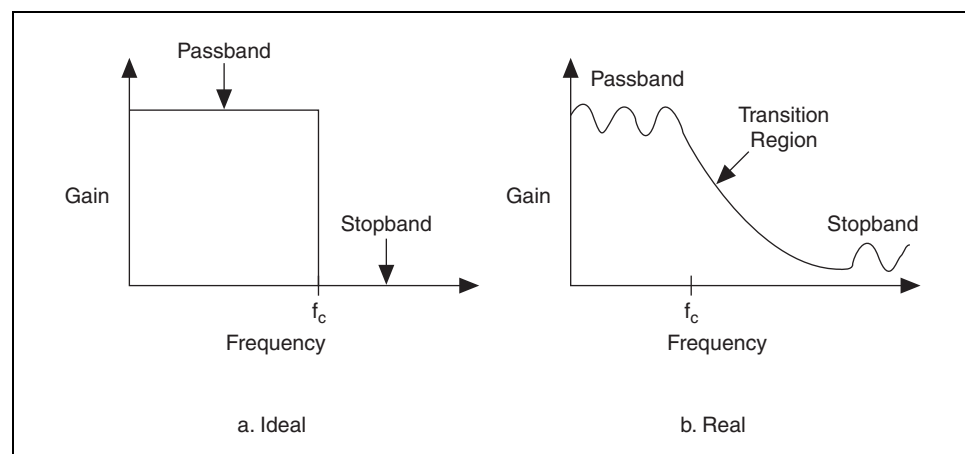
End of Exercise 5-2

D. Filtering

Signal conditioning systems can include filters to reject unwanted noise within a certain frequency range. Almost all DAQ applications are subject to some degree of 50 or 60 Hz noise picked up from the power lines or machinery. Therefore, most signal conditioning systems include lowpass filters designed specifically to provide maximum rejection of 50 or 60 Hz noise. For example, the SCXI-1125 module includes a lowpass filter with a cutoff bandwidth of 4 Hz so that rejection of 50 or 60 Hz noise is maximized (90 db).

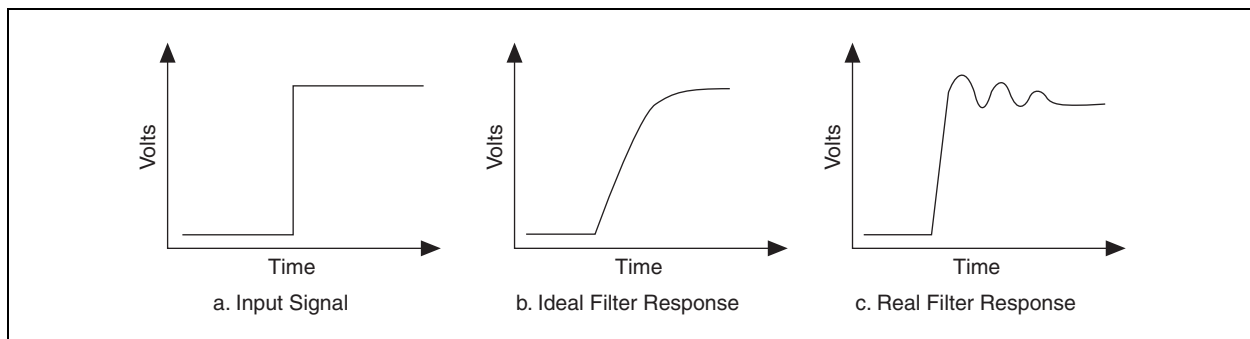
Filters are generally grouped into one of five classifications—lowpass, highpass, bandpass, bandstop, and all-pass. These classifications refer to the frequency range (the passband) of signals that the filter is intended to pass from the input to the output without attenuation. Because most National Instruments signal conditioning modules use a lowpass filter, this section focuses on lowpass filters.

An ideal lowpass filter does not attenuate any input signal frequency components in the passband, which is defined as all frequencies below the cutoff frequency. An ideal lowpass filter completely attenuates all signal components in the stopband, which includes all frequencies above the cutoff frequency. The ideal lowpass filter also has a phase shift that is linear with respect to frequency. This linear phase property means that signal components of all frequencies are delayed by a constant time, independent of frequency, thereby preserving the overall shape of the signal. Real filters subject input signals to mathematical transfer functions that approximate the characteristics of an ideal filter. The following illustration compares the attenuation of transfer functions of a real filter and an ideal filter.



As this illustration shows, a real filter has ripple (an uneven variation in attenuation versus frequency) in the passband, a transition region between the passband and the stopband, and a stopband with finite attenuation and ripple.

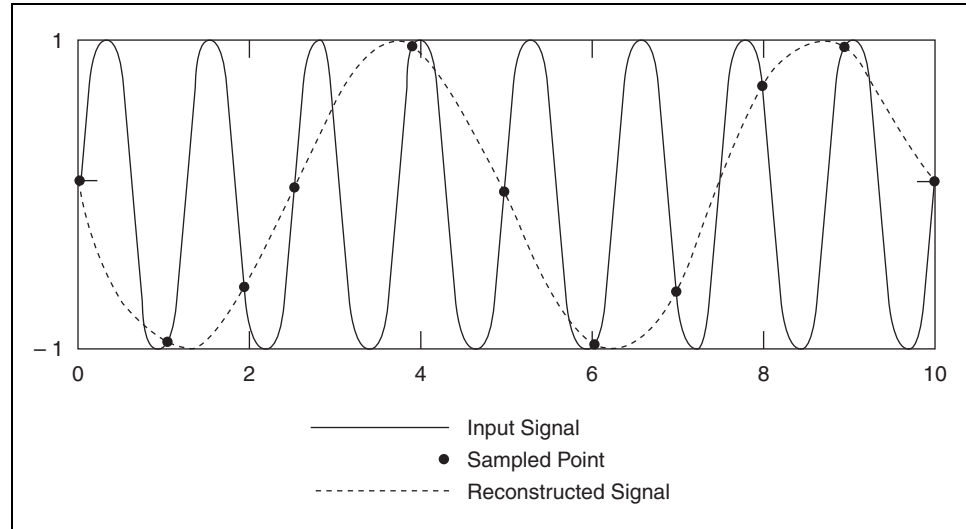
In addition, real filters have some nonlinearity in their phase response, which causes signal components at higher frequencies to be delayed by longer times than signal components at lower frequencies, resulting in an overall shape distortion of the signal. This can be observed when a square wave or step input is sent through a lowpass filter. An ideal filter smooths the edges of the input signal. A real filter causes some ringing in the total signal because the higher-frequency components of the signal are delayed. The following illustration shows examples of these responses to a step input.



Anti-aliasing Filters

Another common use of filters is to prevent signal aliasing—a phenomenon that arises when a signal is undersampled (sampled too slowly). The Nyquist theorem states that when you sample an analog signal, any signal components at frequencies greater than one-half the sampling frequency appear in the sampled data as a lower frequency signal. You can avoid this signal distortion only by removing any signal components above one-half the sampling frequency with lowpass filters before the signal is sampled.

The following illustration shows a sine wave signal sampled at the indicated points. When these sampled data points are used to reconstruct the waveform, as shown by the dotted line, the signal appears to have a lower frequency than the original sine wave.

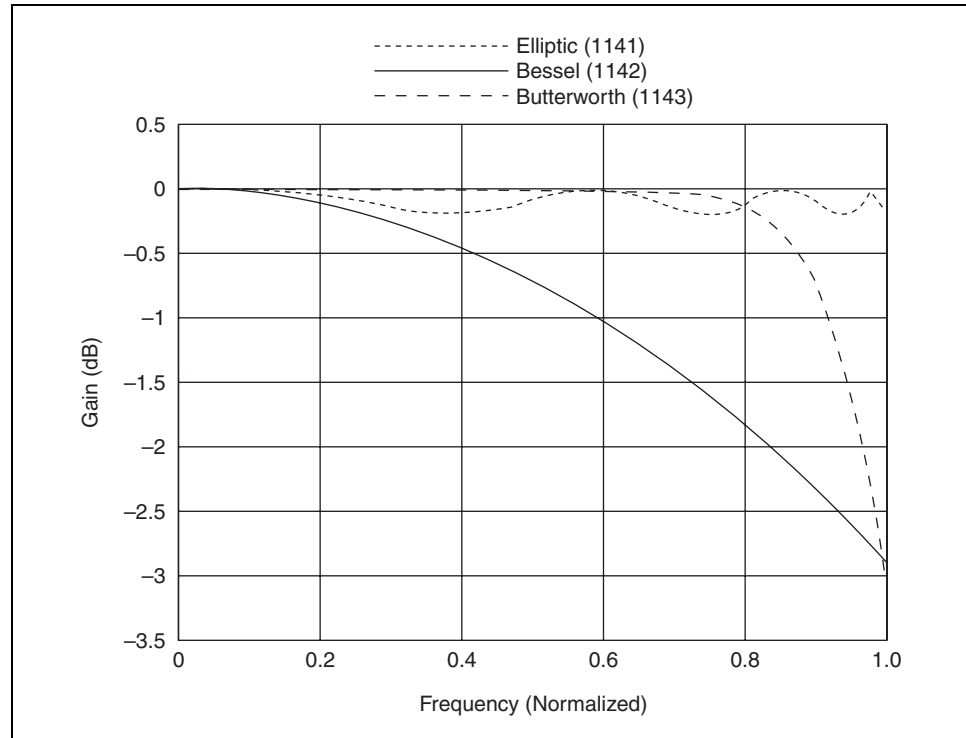


You can increase the sampling rate or pass the signal through a lowpass filter to remove high-frequency components. Increasing the sampling rate can be expensive and often impractical, especially when the upper limit bandwidth of high-frequency noise can be much greater than the bandwidth of the signal of interest. Therefore, a common practice is to use lowpass filters to remove any frequency elements above the Nyquist frequency.

Only analog filters can prevent aliasing. Digital filters cannot remove aliased signals because it is impossible to remove aliasing after the signal has been sampled.

Typically, a programmable analog filter is used as a front-end signal conditioner for digitizing instruments such as DAQ devices or modules. Analog filters are available with standard transfer functions that provide trade-offs in real filter characteristics, such as rolloff, passband ripple, and phase linearity. Standard transfer functions include Butterworth, Chebyshev, Bessel, and elliptic filters. For example, Butterworth filters exhibit very flat frequency response in the passband, while Chebyshev filters provide steeper attenuation at the expense of some passband ripple. The Bessel filter provides a linear phase response over the entire passband, minimizing distortion of waveshapes but exhibits a less sharp rolloff and, therefore, less attenuation in the stopband. The Causer elliptic filter, with its extremely sharp rolloff, is especially useful as an anti-aliasing filter for multichannel digitizing DAQ systems. However, the large phase nonlinearity makes it more appropriate for applications involving analysis of the frequency content of signals as opposed to phase content or waveform shape.

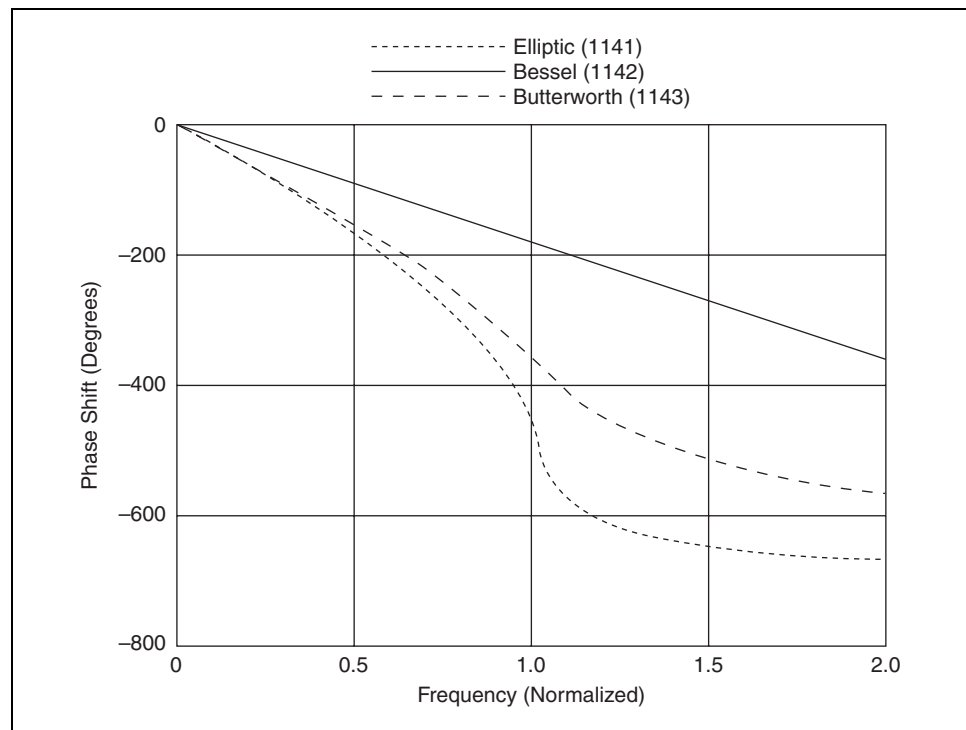
The following illustration shows the transfer function of three signal conditioning modules used by National Instruments. The elliptic filter, with its extremely sharp rolloff, is especially useful as an anti-aliasing filter for multichannel digitizing DAQ systems.



The excellent attenuation performance of the elliptic filter above the cutoff frequency proves beneficial as an anti-aliasing filter. You can sample at a lower rate because of the sharper rolloff of the filter. For example, consider a 16-channel application with the signals of interest limited to a 10 kHz bandwidth. However, anti-aliasing filters are required to prevent distortion by unwanted noise signals above 10 kHz. With the SCXI-1141 eighth-order elliptic filters, for example, you can program the filter cutoff frequency for the 16 channels for 10 kHz. With the cutoff frequency set to 10 kHz, the attenuation of the SCXI-1141 filters reaches 80 dB at about 15 kHz. Therefore, the sampling rate of the digitizing DAQ device can safely be set to twice the 15 kHz frequency, or 30 kS/s. Because 16 channels are sampled by the one DAQ device, the device is programmed for an aggregate sampling rate of 480 kS/s.

If you use a different type of filter with a less sharp rolloff, the DAQ device would need a much higher sampling rate. For example, a typical eighth-order Butterworth filter programmed for a cutoff frequency of 10 kHz reaches an attenuation of 80 dB at about 30 kHz. The sampling rate of the DAQ device used with this filter must be set at 60 kS/s per channel, or 960 kS/s aggregate. This higher sampling rate requires a more expensive DAQ device (or fewer channels per DAQ device), and greater data storage capacity. Therefore, the sharper attenuation of the elliptic filter reduces the requirements of the digitizer, allowing lower sampling rates and lower data storage requirements.

However, the elliptic filter has a large phase nonlinearity as shown in the following illustration. This nonlinear phase response of the filter makes the filter more appropriate for applications that involve analysis of frequency content as opposed to phase content or waveform shape.



When you choose a data acquisition filter, consider both the magnitude and phase response of the filter for a particular application to ensure no loss of signal information occurs as a result of filtering.

Traditional analog filters are typically built with op-amps, resistors, and capacitors. However, switched-capacitor technology is commonly used in the implementation of adjustable anti-aliasing filters. Essentially, a switched capacitor replaces the resistor in the more traditional analog filter designs. Because the impedance of the switched capacitor is a function of the switching frequency, you can vary the cutoff frequency of the switched

capacitor filter by varying the frequency of the clock signal that controls switching. Also, it is easier to accurately fabricate filters composed of op-amps and matched capacitors.

However, switched-capacitor filters have potential drawbacks. Used alone, switched-capacitor filters tend to be too noisy for analog conditioning applications. Therefore, the NI signal conditioning modules use a unique hybrid design of switched-capacitor filters and continuous-time filters to combine the technological and performance benefits of both. The main components of an NI signal conditioning filter stage consist of the switched-capacitor discrete-time filter, the programmable analog continuous-time prefilter, and the programmable analog continuous-time postfilter.

- **Switched-capacitor, discrete-time filter**—The cutoff frequency is easily controlled by varying the frequency of the input clock signal.
- **Programmable analog continuous-time prefilter**—Switched-capacitor filters are sampling devices and are subject to aliasing. Therefore, the analog prefilter attenuates any frequencies outside the Nyquist limit of the switching frequency.
- **Programmable analog continuous-time postfilter**—The output of the switched-capacitor filter, a stairstep representation of the analog signal, is reconstructed with the analog postfilter. This postfilter also removes any feedthrough noise from the high-frequency clock that drives the switched-capacitor filter.

Exercise 5-3 Using Hardware Filters to Prevent Signal Aliasing

Objective: To use a hardware filter as an anti-aliasing filter.

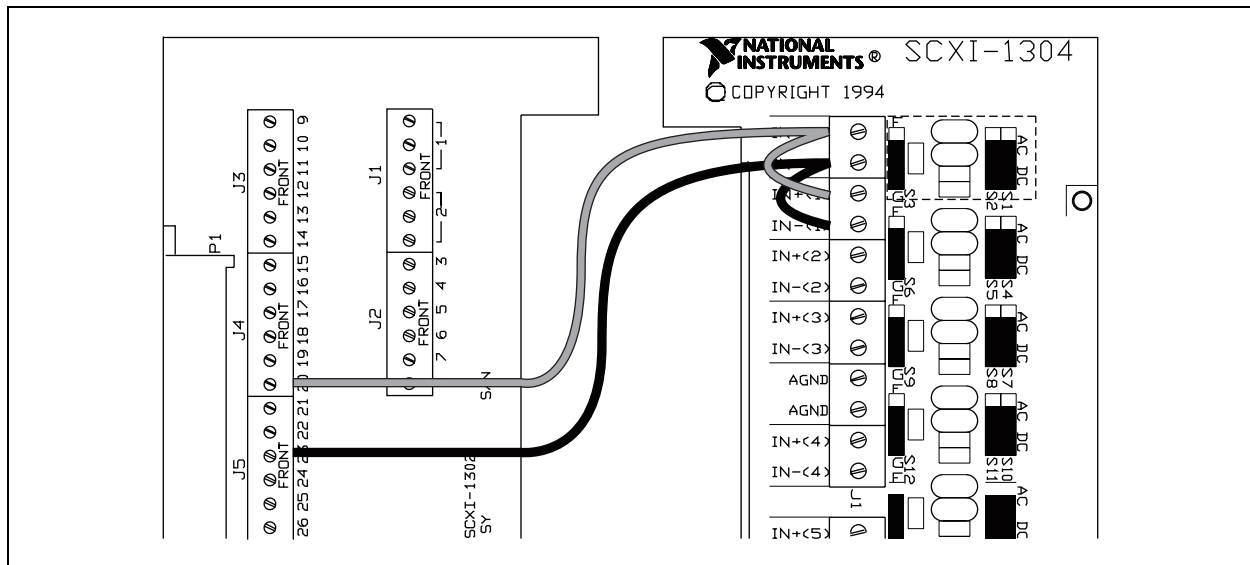
In this exercise, you programmatically apply filter settings to the SCXI-1141 Lowpass Filter Module through LabVIEW and show the benefits of using hardware filtering to prevent aliasing of input signals.

Connect the Analog Output of the DAQ Device



Caution SCXI modules are *not* hot-swappable. Make sure the chassis is powered off before removing or adding any modules, terminal blocks, or cables.

1. Remove the covers from the SCXI-1302 and SCXI-1304 terminal blocks.
2. Make sure you have wires long enough to run from the SCXI-1302 to the SCXI-1304 and make the connections as shown in the following illustration.



- a. Look at the SCXI-1304 and find the AC/DC coupling switches [S1/S2 for IN (0) and S4/S5 for IN (1)] and make sure they are set to the DC setting as shown in the previous illustration.
- b. Look at the SCXI-1304 and find the Floating and Ground Referenced Signal Configuration switches [S3 for IN (0) and S6 for IN (1)] and make sure they are in the Ground Referenced setting as shown in the previous illustration.

- c. Connect a wire to Screw Terminal 20 of the SCXI-1302 (DAC0OUT pin 22 of the DAQ device) to IN +(0) of the SCXI-1304.
 - d. Connect a smaller piece of wire from IN +(0) to IN +(1) of the SCXI-1304.
 - e. Connect a wire to screw terminal 23 of the SCXI-1302 (AOGND pin 55 of the DAQ device) to IN -(0) of the SCXI-1304.
 - f. Connect a smaller piece of wire from IN -(0) to IN -(1) of the SCXI-1304.
3. Reattach the covers to their respective terminal blocks.
 4. Attach the SCXI-1302 to the SCXI-1180 feedthrough panel.
 5. Attach the SCXI-1304 to the SCXI-1141 lowpass filter module.
 6. Power on the SCXI chassis.
 7. Return to MAX. Right-click the DAQ device under **NI-DAQmx Devices** and select **Test Panels** from the shortcut menu.
 8. Select the **Analog Output** tab and change the following settings:
 - **Channel Name:** DevX/ao0
 - **Output Mode:** Sine Generation
 - **Update Rate:** 1000.0
 - **Transfer Mechanism:** DMA
 - **Output Voltage/Amplitude:** 5.00Click the **Start** button.
 9. Select the **Analog Input** tab and change the following settings:
 - **Channel Name:** SC1Mod4/ai0
 - **Acquisition Mode:** Continuous
 - **Max Input Limit:** 5.00
 - **Min Input Limit:** -5.00
 10. Click the **Start** button. You should see a sine wave.
 11. Change the **Channel Name** to read the signal on channel 1 of the SCXI-1141. You should see the same sine wave as on channel 0.

MAX Configuration

1. Create two new NI-DAQmx channels for the two channels you tested on the SCXI-1141. Name the channels `FilterChannel1` and `FilterChannel2`.

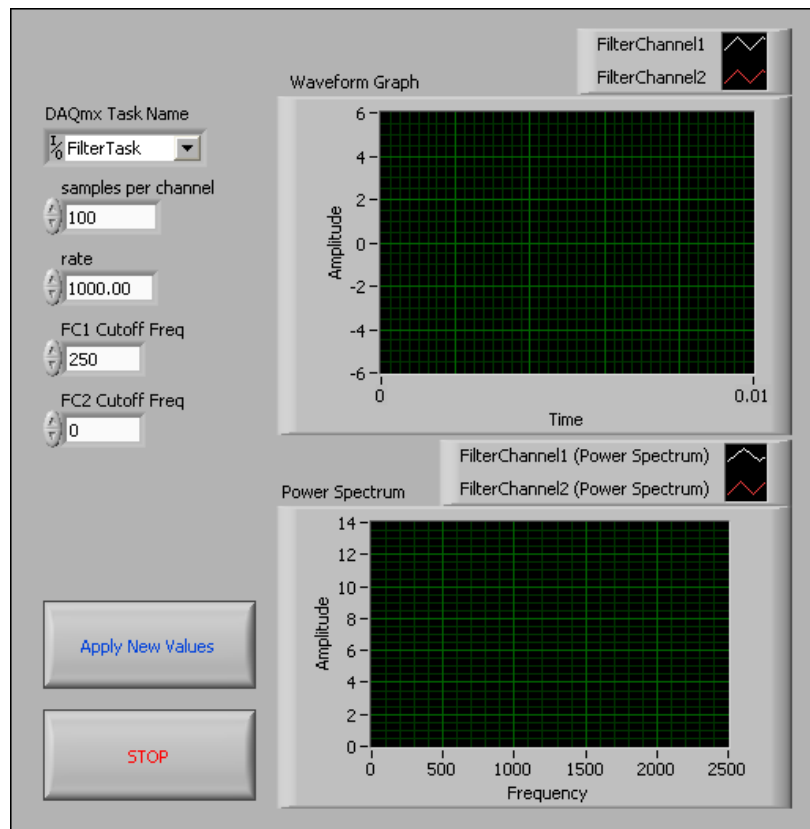
For each channel, select the **Device** tab, select **Lowpass Filter Enable**, and set the cutoff frequency to 10000.

2. Add `FilterChannel1` and `FilterChannel2` to a new task called `FilterTask`.
3. Save the task.

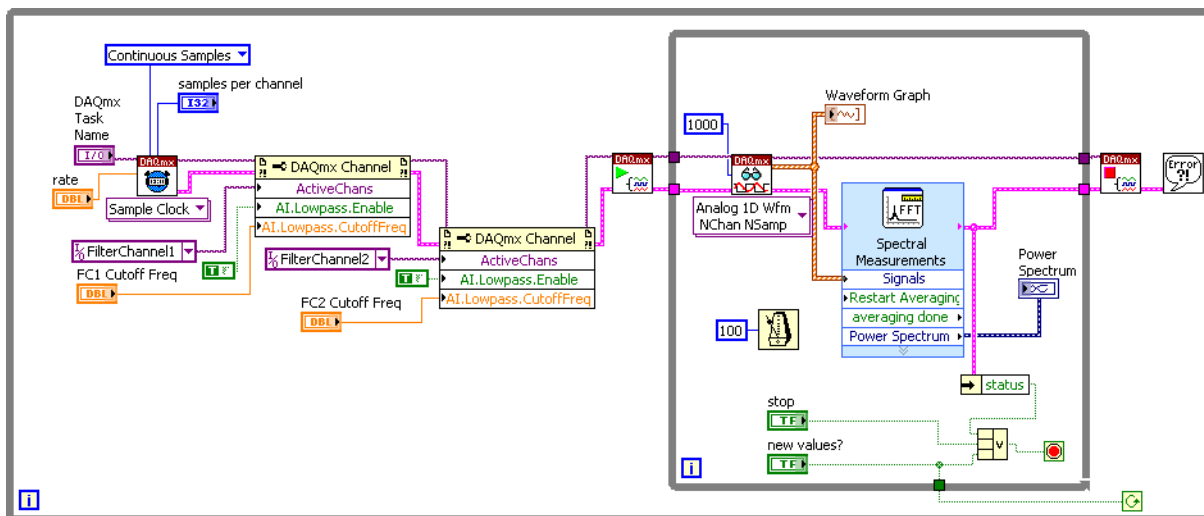
Configuring SCXI Filter Settings Programmatically in LabVIEW

Complete a VI that acquires sine wave data generated by the DAQ device using the SCXI-1141 lowpass filter module. This VI demonstrates how to programmatically set the filters of the SCXI-1141 and how hardware filtering prevents aliasing on the input signal.

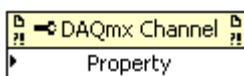
1. Launch LabVIEW and open the SCXI-1141 Hardware Filtering VI located in the `C:\Exercises\LabVIEW DAQ` directory. The following front panel displays.



2. Complete the block diagram shown in the following figure.



a. Place the DAQmx Timing VI, located on the **Functions»All Functions»NI Measurements»DAQmx - Data Acquisition** palette, on the block diagram. This VI configures the sample timing rate and sampling mode. Right-click the **sample mode** input and select **Create»Constant** from the shortcut menu.



b. Place the DAQmx Channel Property Node, located on the **Functions»All Functions»NI Measurements»DAQmx - Data Acquisition** palette, on the block diagram. Use this property node to enable lowpass filtering and set the cutoff frequency.



c. Place the DAQmx Start Task VI, located on the **Functions»All Functions»NI Measurements»DAQmx - Data Acquisition** palette, on the block diagram. This VI starts a task operation.



d. Place the DAQmx Read VI, located on the **Functions»All Functions»NI Measurements»DAQmx - Data Acquisition** palette, on the block diagram. Use this polymorphic VI to read sample data. Select the **Analog»Multiple Channels»Multiple Samples»1D Waveform** instance from the pull-down menu.



e. Place the Spectral Measurements Express VI, located on the **Functions»Signal Analysis** palette, on the block diagram. This Express VI configures the analysis parameters for spectral measurements of a signal. In the **Configure Spectral Measurements** dialog box that displays, configure the following settings:

- (1) Set **Spectral Measurement** to **Power Spectrum** and **Result** to **Linear** mode.
- (2) Select **Hanning** from the **Window** pull-down menu and place a checkmark in the **Averaging** checkbox.

- (3) Accept the default values for Averaging.
- (4) Click the **OK** button to close the dialog box.

Connect the **Power Spectrum** output to the Power Spectrum Graph to display the frequency of the input signal and the effects of aliasing.



- f. Place the DAQmx Stop Task VI, located on the **Functions»All Functions»NI Measurements»DAQmx - Data Acquisition** palette, on the block diagram. This VI stops a task operation.



- g. Place the Wait until Next ms Multiple function, located on the **Functions»All Functions»Time & Dialog** palette, on the block diagram. This function controls the While Loop execution rate. Set the millisecond multiple to 100.



- h. Place the Simple Error Handler VI, located on the **Functions»All Functions»Time & Dialog** palette, on the block diagram. In the event of an error, this VI displays a dialog box with information about the error and where it occurred.



- i. Place the Unbundle by Name function, located on the **Functions»All Functions»Cluster** palette, on the block diagram. This function returns the error **status** Boolean value. If an error occurs, the execution of the While Loop stops.



- j. Place the Compound Arithmetic function, located on the **Functions»Arithmetic & Comparison»Express Numeric** palette, on the block diagram. This function performs arithmetic on one or more numeric, array, cluster, or Boolean inputs. The connector pane displays the default data types for this polymorphic function. Right-click the function and select **Change Mode»OR** from the shortcut menu. You can resize this function to have more than two inputs.

3. Save the VI.
4. Return to the front panel and set the front panel controls with the following values:
 - **Samples per channel:** 100
 - **Rate:** 10000.00
 - **FC1 Cutoff Freq:** 500
 - **FC2 Cutoff Freq:** 0 (Setting the filter to 0 bypasses the filter on the SCXI-1141 Module.)
5. Select **Help»Find Examples** to open the NI Example Finder and browse by task to **Hardware Input and Output»DAQmx»Analog Generation»Voltage**. Open the Cont Gen Voltage Wfm-Int Clk-Variable Rate VI. Run the VI and generate a 100 Hz sine wave on analog output channel 0.

6. Run the SCXI-1141 Hardware Filtering VI.
7. Adjust the **Analog Output Frequency** control from 100 Hz to around 5,000 Hz and notice how the two signals displayed on the graphs of the SCXI-1141 Hardware Filtering VI change. Notice the rolloff of channel 0 around 500 Hz on the Power Spectrum graph. As you continue to increase the sine wave frequency, watch channel 1 on the Power Spectrum graph and notice what frequency it reads compared to what you are outputting.
8. Adjust **CH 0 Cutoff Frequency** to 2500.00 and click **Apply New Values**. Repeat step 7.



Note The effect you are witnessing is aliasing, a false lower frequency component that appears in sampled data acquired at too low a sampling rate compared to the Nyquist Frequency. Once aliasing has been introduced into a sampled signal, there is no standard way to remove it. By introducing an anti-aliasing filter (in this instance, the SCXI-1141) before the ADC of the DAQ device, you restrict the bandwidth of the input signal to meet the Nyquist sampling criteria. The Nyquist theorem states that a signal must be sampled at greater than twice the highest frequency component of the signal to accurately represent the frequency domain waveform. Otherwise, the high-frequency content aliases at a frequency inside the spectrum of interest.

9. Experiment with the cutoff frequency settings of channel 0 and channel 1 and click **Apply New Values** to set different filter settings to the SCXI-1141.
10. Stop and close both VIs.

End of Exercise 5-3

E. Isolation

Improper grounding of the system is one of the most common causes of measurement problems, noise, and damaged DAQ devices. Signal conditioning systems with isolation can prevent most of these problems. Such devices pass the signal from its source to the measurement device without a physical connection by using a transformer, optical, or capacitive coupling techniques. Besides breaking ground loops, isolation blocks high-voltage surges and rejects common-mode voltage, protecting operators and expensive measurement equipment.

Suppose you need to monitor temperature using thermocouples soldered to a high-voltage machine that radiates large electro-magnetic fields. Although the thermocouples output a differential voltage of less than 50 mV, this output voltage can be at a high output potential with respect to ground due to the capacitive coupling that the machine has with the thermocouple. This potential between both leads of a differential signal and ground is called the common-mode voltage. Ideally it should be completely ignored by the measurement system. Connecting the thermocouple leads directly to a nonisolated device, which can typically handle 12 V of common-mode voltage, would probably damage the device. However, you can connect the thermocouple leads to an isolated signal conditioner, which rejects the high common-mode voltage, safely passing the 50 mV differential signal on to the measurement device for an accurate measurement.

Isolation Specifications

Manufacturers specify isolation differently. Some manufacturers just provide an isolation number without describing if that number is a signal level or a transient level. Without this information and an acute understanding of the I/O signals, you can damage the measurement system and possibly create a shock hazard for the system operators. Agencies such as Underwriters Laboratories (UL) and the International Electrotechnical Commission (IEC) designate compliance requirements for safe design of high voltage instrumentation. Products that display these symbols are tested, in some cases by the agency themselves, to ensure they meet their specifications.

In addition to looking for the seal of approval from one of these agencies, the safest way to determine the isolation rating of a signal conditioning system is to look for two key specifications—working voltage rating and installation category rating.

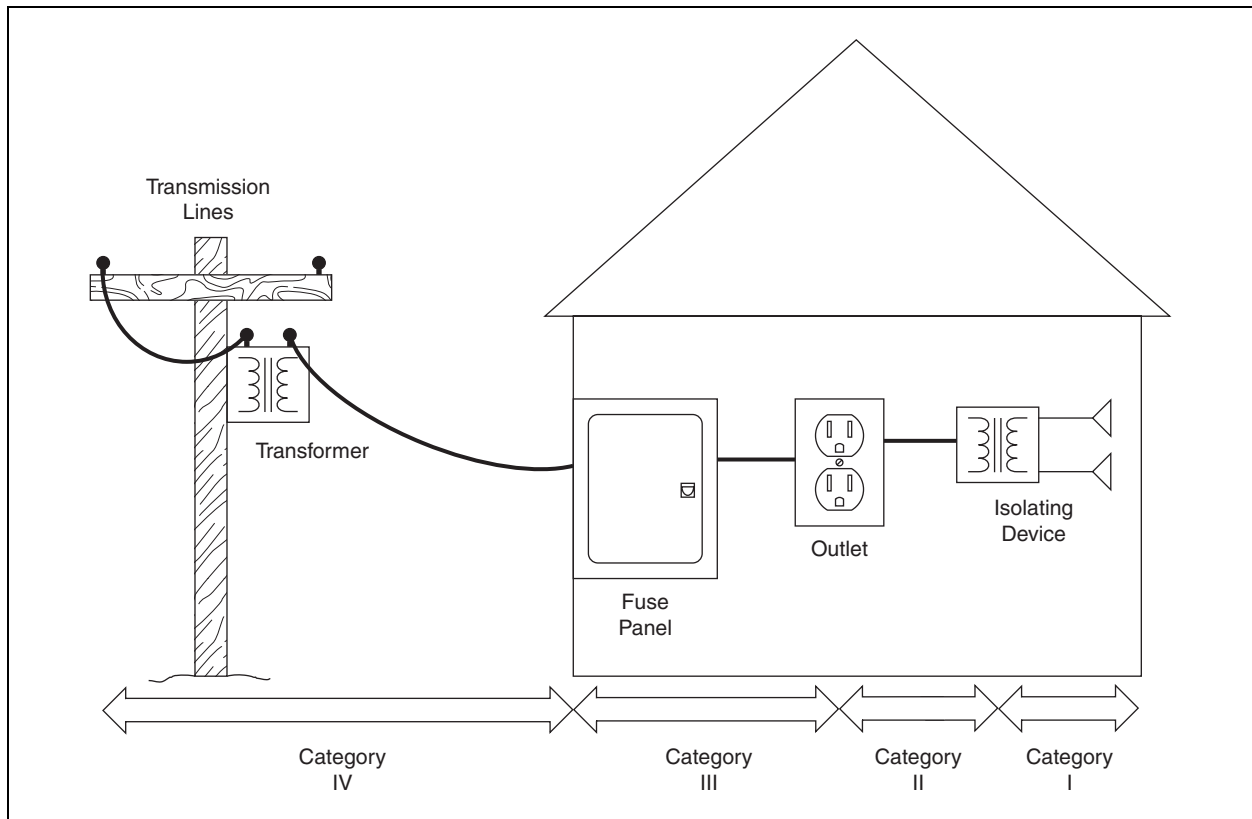
Working Voltage Rating

This specification describes the maximum continuous voltage that you can apply to inputs under normal working conditions. This specification is described with respect to the unit ground reference and includes both the signal level and any common-mode voltage associated with the signal.

Installation Category Rating

By definition, an installation category rating describes the locations where you can use a particular measurement device based on the possible transient signals at that location. In a more general sense, this specification describes the possible transient signals that the device can withstand.

Installation category ratings are described as Category I through Category IV, as shown in the following illustration.



Depending on the location of the electrical distribution system, there is a certain amount of damping in the system. Damping occurs less toward the generating plant and increases as the transmission system spreads out. This damping reduces the overvoltages that are present in the system. The closer you are to the source, the higher the anticipated voltage transients. Depending on the location of the measurement system, certain precautions need to be made to protect the measurement system from potentially dangerous overvoltages that are present in the distribution system. The IEC created the following four categories to partition circuits with different levels of overvoltages.

- Installation Category IV—Distribution level. Equipment such as generators, substations, and transformers.
- Installation Category III—Fixed installation. Equipment permanently connected to the distribution network, such as air conditioners and furnaces.
- Installation Category II—Equipment consuming energy from a fixed installation system. This includes equipment such as drills, televisions, radios, and computers.
- Installation Category I—Equipment for connection to circuits where transient overvoltages are limited to a sufficiently low level by design. Category I equipment includes equipment such as low-voltage power supplies.

Safety isolation provides a safety barrier between the user and their equipment and high voltages and overvoltage transients. Equipment that provides safety isolation is a very desirable feature.

All isolated National Instruments SCXI modules are double insulated for a continuous working voltage of $250 V_{\text{rms}}$. In addition, they are tested by applying a 2,300 V source to their inputs for a minute to make sure that the module does not fail with overvoltage transients. The isolated SCXI modules adhere to the IEC-1010 specification for a Category II installation.

Refer to the following IEC publications for more information about installation categories.

- IEC 664-1—Installation coordination for equipment within low-voltage systems
- IEC 1010-1—Safety requirements for electrical equipment for measurement, control, and laboratory use.

F. Transducer Conditioning

Transducers are devices that convert physical phenomena such as temperature, strain, pressure, or light into electrical properties such as voltage or resistance. Transducer characteristics define many of the signal conditioning requirements of a DAQ system.

G. Thermocouples

One of the most frequently used temperature transducers is the thermocouple. Thermocouples are very rugged and inexpensive and can operate over a wide temperature range. A thermocouple is created whenever two dissimilar metals touch, and the contact point produces a small open-circuit voltage as a function of temperature. This thermoelectric voltage is known as the Seebeck voltage, named after Thomas Seebeck, who discovered it in 1821. The voltage is nonlinear with respect to temperature. However, for small changes in temperature, the voltage is approximately linear, or

$$\Delta V \approx S\Delta T$$

where ΔV is the change in voltage, S is the Seebeck coefficient, and ΔT is the change in temperature.

S varies with changes in temperature, which causes the output voltages of thermocouples to be nonlinear over their operating ranges. Several types of thermocouples are available and are designated by capital letters that indicate their composition according to American National Standards Institute (ANSI) conventions. For example, a J-type thermocouple has one iron conductor and one constantan (a copper-nickel alloy) conductor.

You can monitor thermocouples with versatile PC-based data acquisition systems. Thermocouples have some special signal conditioning requirements.

Thermocouple Circuits

To measure a thermocouple Seebeck voltage, you cannot simply connect the thermocouple to a voltmeter or other measurement system, because connecting the thermocouple wires to the measurement system creates additional thermoelectric circuits.

Consider the circuit illustrated in Figure 5-1, in which a J-type thermocouple is in a candle flame that has a temperature you want to measure. The two thermocouple wires are connected to the copper leads of a DAQ device.

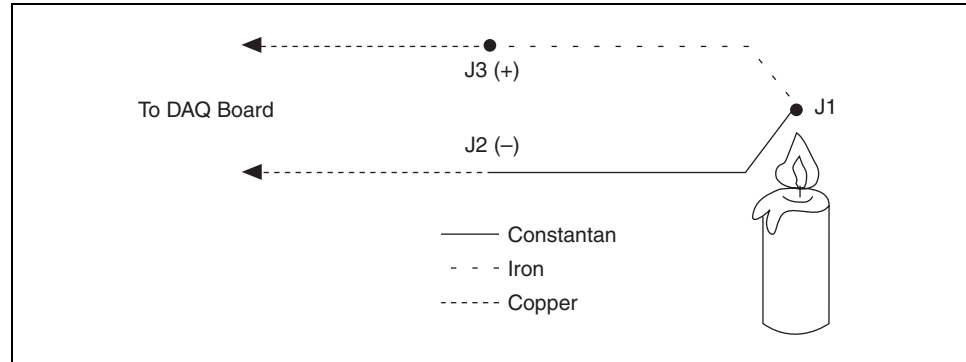


Figure 5-1. J-Type Thermocouple

Notice that the circuit contains three dissimilar metal junctions—J1, J2, and J3. J1, the thermocouple junction, generates a Seebeck voltage proportional to the temperature of the candle flame. J2 and J3 each have their own Seebeck coefficient and generate their own thermoelectric voltage proportional to the temperature at the DAQ terminals. To determine the voltage contribution from J1, you need to know the temperatures of junctions J2 and J3 and the voltage-to-temperature relationships for these junctions. You can then subtract the contributions of the parasitic thermocouples at J2 and J3 from the measured voltage.

Cold-Junction Compensation

Thermocouples require some form of temperature reference to compensate for parasitic thermocouples. The term cold junction comes from the traditional practice of holding this reference junction at $0\text{ }^{\circ}\text{C}$ in an ice bath. The National Institute of Standards and Technology (NIST) thermocouple reference tables are created with this setup, illustrated in Figure 5-2.

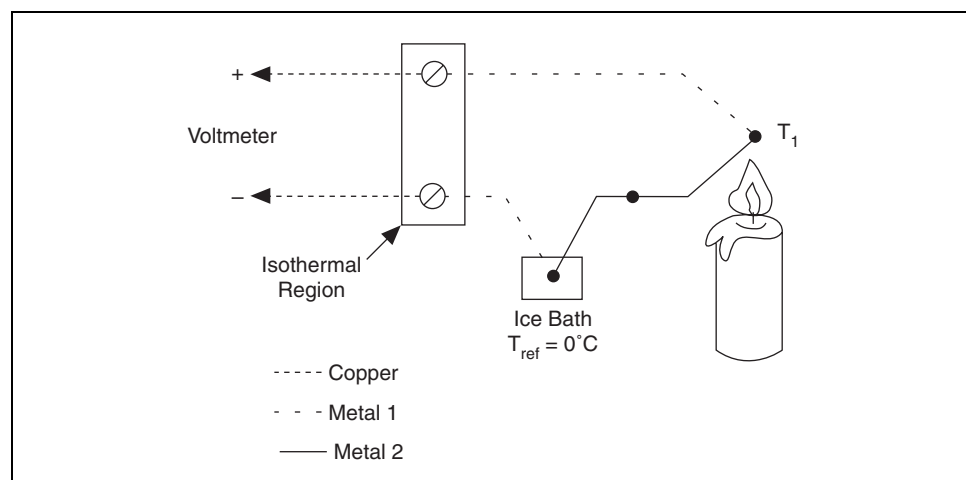


Figure 5-2. Traditional Temperature Measurement with Reference Junction Held at $0\text{ }^{\circ}\text{C}$

In Figure 5-2, the measured voltage depends on the difference in temperatures T_1 and T_{ref} . In this case, T_{ref} is $0\text{ }^\circ\text{C}$. Notice that because the voltmeter lead connections are the same temperature, or isothermal, the voltages generated at these two points are equal and opposing. Therefore, the net voltage error added by these connections is zero.

Under these conditions, if the measurement temperature is above $0\text{ }^\circ\text{C}$, a thermocouple has a positive output. If the measurement temperature is below $0\text{ }^\circ\text{C}$, the output is negative. When the reference junction and the measurement junction are the same temperature, the net voltage is zero.

Although an ice bath reference is accurate, it is not always practical. A more practical approach is to measure the temperature of the reference junction with a direct-reading temperature sensor and subtract the parasitic thermocouple thermoelectric voltage contributions. This process is called cold-junction compensation (CJC). You can simplify computing CJC by taking advantage of some thermocouple characteristics.

By using the Thermocouple Law of Intermediate Metals and making some simple assumptions, you can see that the voltage the DAQ device measures in Figure 5-1 depends only on the thermocouple type, the thermocouple voltage, and the cold-junction temperature. The measured voltage is in fact independent of the composition of the measurement leads and the cold junctions, J2 and J3.

According to the Thermocouple Law of Intermediate Metals, illustrated in Figure 5-3, inserting any type of wire into a thermocouple circuit has no effect on the output as long as both ends of that wire are the same temperature, or isothermal.

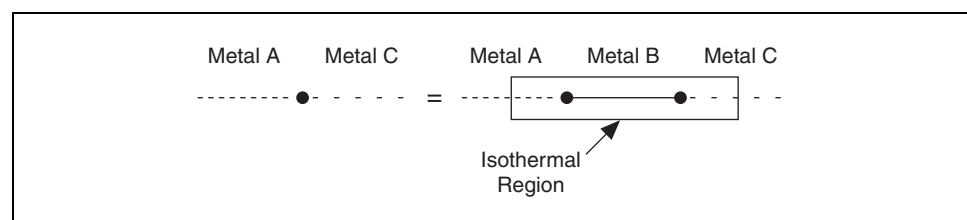


Figure 5-3. Thermocouple Law of Intermediate Metals

Consider the circuit in Figure 5-4. This circuit is similar to the previously described circuit in Figure 5-1, but a short length of constantan wire has been inserted just before junction J3, and the junctions are assumed to be held at identical temperatures. Assuming that junctions J3 and J4 are the same temperature, the Thermocouple Law of Intermediate Metals indicates that the circuit in Figure 5-4 is electrically equivalent to the circuit in Figure 5-1. Consequently, any result taken from the circuit in Figure 5-4 also applies to the circuit illustrated in Figure 5-1.

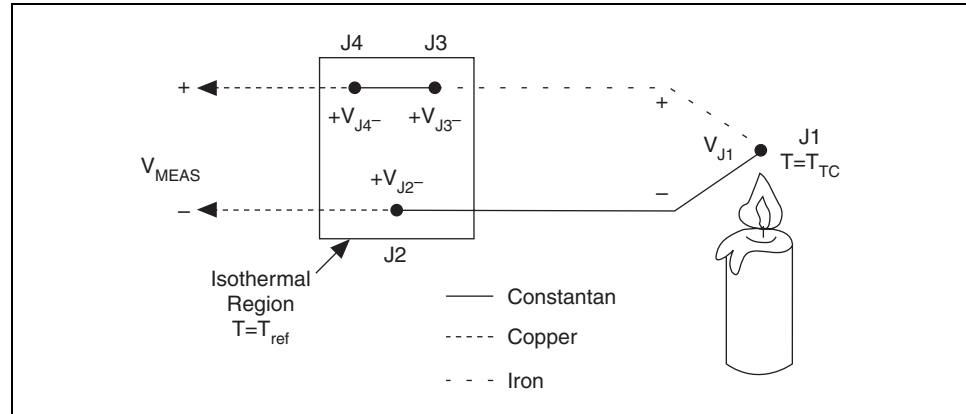


Figure 5-4. Inserting an Extra Lead in the Isothermal Region

In Figure 5-4, junctions J2 and J4 are the same type (copper-constantan). Because both are in the isothermal region, J2 and J4 are also the same temperature. The junctions occur in opposite directions, however, so their total contribution to the measured voltage is zero. Junctions J1 and J3 are both iron-constantan junctions and also point in opposite directions but may be different temperatures. Therefore, junctions J1 and J3 are the only two junctions with outputs that have any effect on the total voltage measured.

Using the notation $V_{J_x}(T_y)$ to indicate the voltage generated by the junction J_x at temperature T_y , the general thermocouple problem is reduced to the following equation:

$$V_{MEAS} = V_{J1}(T_{TC}) + V_{J3}(T_{ref}) \quad (5-1)$$

where V_{MEAS} is the voltage the DAQ device measures, T_{TC} is the temperature of the thermocouple at J1, and T_{ref} is the temperature of the reference junction.

Notice that in Equation 5-1, $V_{J_x}(T_y)$ is a voltage generated at temperature T_y with respect to some reference temperature. As long as both V_{J1} and V_{J3} are functions of temperature relative to the same reference temperature, Equation 5-1 is valid. As stated earlier, for example, NIST thermocouple reference tables are generated with the reference junction held at 0 °C.

Because junction J3 is the same type as J1 but in the opposite direction, $V_{J3}(T_{ref}) = -V_{J1}(T_{ref})$. Because V_{J1} is the voltage that the thermocouple type undergoing testing generates, this voltage can be renamed V_{TC} . Therefore, Equation 5-1 is rewritten as follows:

$$V_{MEAS} = V_{TC}(T_{TC}) - V_{TC}(T_{ref}) \quad (5-2)$$

Therefore, by measuring V_{MEAS} and T_{ref} and knowing the voltage-to-temperature relationship of the thermocouple, you can determine the temperature of the thermocouple.

The techniques for implementing cold-junction compensation require that the temperature at the reference junction be sensed with a direct-reading sensor. A direct-reading sensor has an output that depends only on the temperature of the measurement point. Semiconductor sensors, thermistors, or RTDs are commonly used to measure the reference-junction temperature. For example, several SCXI terminal blocks include thermistors that are located near the screw terminals to which thermocouple wires are connected.



Note NI-DAQ and NI LabVIEW and Measurement Studio include built-in routines that perform the required software compensation.

Linearizing the Data

Thermocouple output voltages are highly nonlinear. The Seebeck coefficient can vary by a factor of three or more over the operating temperature range of some thermocouples. For this reason, you must either approximate the thermocouple voltage-versus-temperature curve using polynomials or use the following look-up table. The polynomials are in the following form:

$$T = a_0 + a_1v + a_2v^2 + \dots + a_nv^n \quad (5-3)$$

where v is the thermocouple voltage in volts, T is the temperature in degrees Celsius, and a_0 through a_n are coefficients that are specific to each thermocouple type. NI software can linearize the thermocouple output voltages for various thermocouples.

Table 5-1. Thermocouple Voltage Output Extremes (mV)

Thermocouple Type	Conductor		Temperature Range (°C)	Voltage Range (mV)	Seebeck Coefficient (µV/°C)
	Positive	Negative			
E	Chromel	Constantan	-270 to 1,000	-9.835 to 76.358	58.70 at 0 °C
J	Iron	Constantan	-210 to 1,200	-8.096 to 69.536	50.37 at 0 °C
K	Chromel	Alumel	-270 to 1,372	-6.548 to 54.874	39.48 at 0 °C
T	Copper	Constantan	-270 to 400	-6.258 to 20.869	38.74 at 0 °C
S	Platinum-10% Rhodium	Platinum	-50 to 1,768	-0.236 to 18.698	10.19 at 600 °C

Table 5-1. Thermocouple Voltage Output Extremes (mV) (Continued)

Thermocouple Type	Conductor		Temperature Range (°C)	Voltage Range (mV)	Seebeck Coefficient (µV/°C)
	Positive	Negative			
R	Platinum-13% Rhodium	Platinum	-50 to 1,768	-0.226 to 21.108	11.35 at 600 °C

Exercise 5-4 Temperature Reading

Objective: To take a temperature measurement with a thermocouple.

In this exercise, you will take a temperature reading with the SCXI-1125 and LabVIEW.

Configuring the SCXI-1125 in MAX

1. Create an NI-DAQmx Channel for the thermocouple input. Use the following settings for the channel:
 - **Measurement Type:** Analog Input
 - **Sensor Type:** Temperature
 - **Temperature Type:** Thermocouple
 - **Physical Channel:** SC1Mod3/ai0
 - **Name:** ThermoTemp
2. In the **Analog Input Thermocouple Channel** dialog box, select the **Settings** tab. Set the input range for 0 to 100 °C. Set the **Thermocouple Type** to J. The **CJC Source** is Built In.



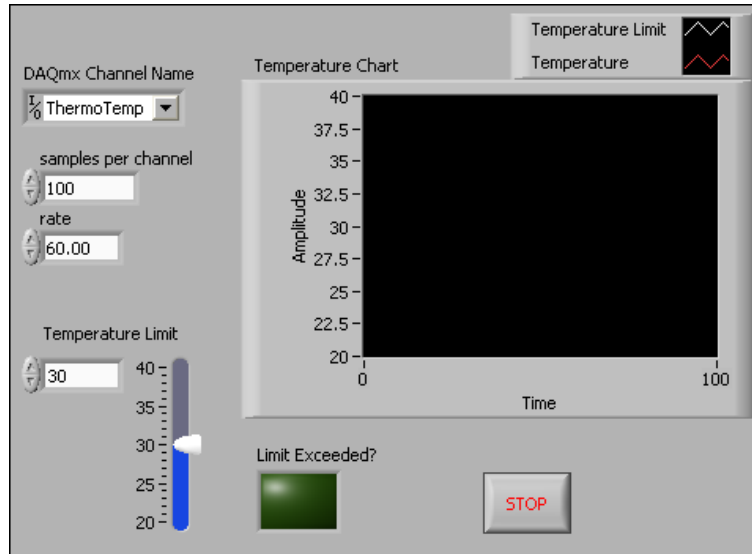
Note Selecting **Built-In** for the CJC source tells MAX and NI-DAQ to use the thermistor built into the SCXI-1327.

3. Select the **Device** tab and set the **Lowpass Filter Cutoff Frequency** to 4 Hz.
4. Click the **Test** button to launch the test panel.
5. Use your fingers or a glass of ice water to test the thermocouple and determine if the reading changes accordingly. If no change is detected or you get an error, inform the instructor. If the virtual channel is working, click the **OK** button and exit MAX.
6. Click the **Yes** button to save the channel.

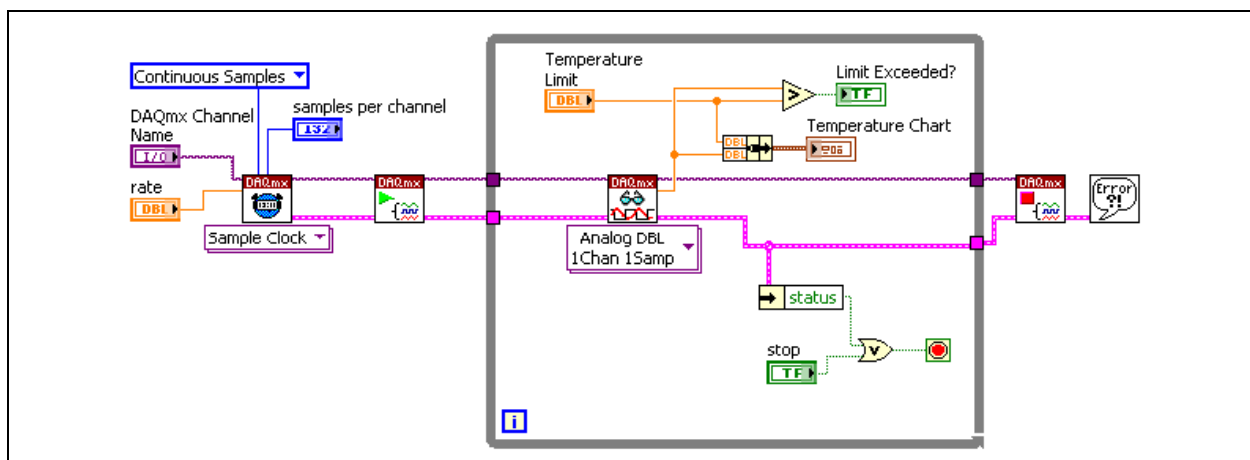
Acquiring Temperature Data in LabVIEW

Complete a VI that acquires temperature data from the thermocouple you connected to the SCXI-1125/SCXI-1327. This VI takes the acquired data, compares it to a user defined limit, plots the data and the limit to a chart, and activates a Boolean indicator if the data exceeds the limit.

1. Open the SCXI-1125 Temperature Reader VI located in the `C:\Exercises\LabVIEW DAQ` directory. The following front panel displays.



2. Complete the following block diagram.



a. Place the DAQmx Timing VI, located on the **Functions»All Functions»NI Measurements»DAQmx - Data Acquisition** palette, on the block diagram. This VI configures the sample timing rate and sampling mode. Right-click the **sample mode** input and select **Create»Constant** from the shortcut menu.



b. Place the DAQmx Start Task VI, located on the **Functions»All Functions»NI Measurements»DAQmx - Data Acquisition** palette, on the block diagram. This VI starts a task operation.



c. Place the DAQmx Read VI, located on the **Functions»All Functions»NI Measurements»DAQmx - Data Acquisition** palette, on the block diagram. This polymorphic VI reads sample data. Select the **Analog»Single Channel»Single Sample»DBL** instance from the pull-down menu.



- d. Place the Bundle By Name function, located on the **Functions»All Functions»Cluster** palette, on the block diagram.
 - e. Place the Simple Error Handler VI, located on the **Functions»All Functions»Time & Dialog** palette, on the block diagram. This VI determines if an error occurred.
 - f. Place the Or function, located on the **Functions»Arithmetic & Comparison»Express Boolean** palette, on the block diagram.
 - g. Place the Unbundle by Name function, located on the **Functions»All Functions»Cluster** palette, on the block diagram. This function returns the error **status** Boolean value. If an error occurs, the While Loop stops.
3. Save the VI.
 4. Set the front panel controls as follows:
 - **DAQmx Channel Name:** `ThermoTemp` (Use the pull-down menu to select temperature instead of typing the name into the control.)
 - **Samples per channel:** 100
 - **Rate:** 60.00
 - **Temperature Limit:** 30
 5. Run the VI.
 6. Use your fingers to heat up the thermocouple and observe as the temperature rises above the limit line displayed on the chart. Adjust the temperature limit up and down and notice the chart and the Boolean indicator.
 7. Stop the VI.
 8. Save and close the VI.

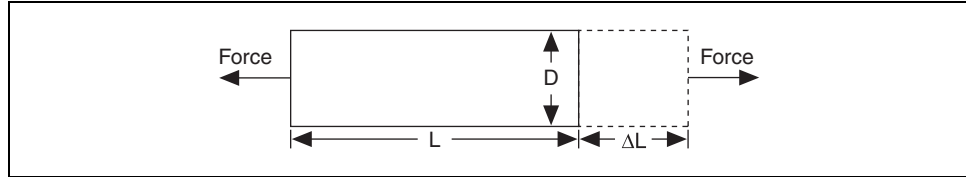
End of Exercise 5-4

H. Strain

Strain (ϵ) is the amount of deformation of a body due to an applied force.

$$\epsilon = \frac{\Delta L}{L}$$

More specifically, strain is defined as the fractional change in length, as shown in the following illustration.



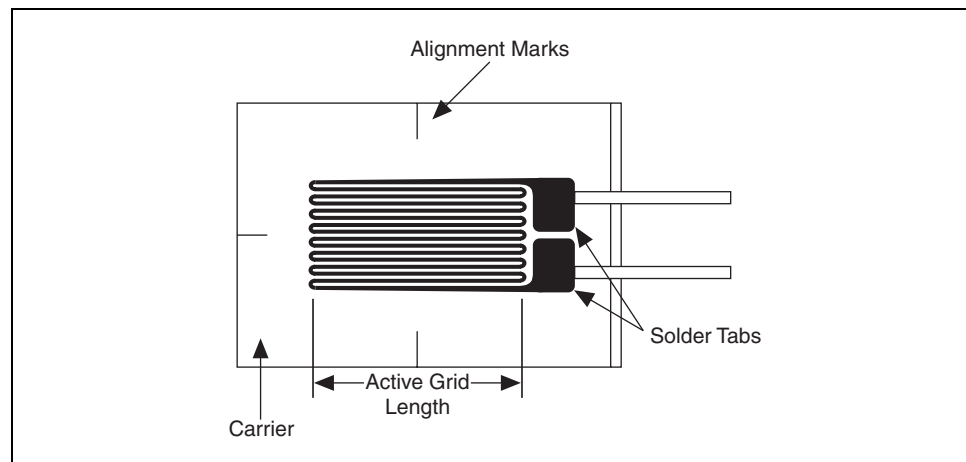
Strain can be positive (tensile) or negative (compressive). Although dimensionless, strain is sometimes expressed in units such as in./in. or mm/mm. In practice, the magnitude of measured strain is very small. Therefore, strain is often expressed as microstrain ($\mu\epsilon$), which is $\epsilon \times 10^{-6}$.

When a bar is strained with a uniaxial force, as in the previous illustration, a phenomenon known as Poisson Strain causes the girth of the bar, D , to contract in the transverse, or perpendicular, direction. The magnitude of this transverse contraction is a material property indicated by its Poisson's Ratio. The Poisson's Ratio ν of a material is defined as 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), or $\nu = -\epsilon_T/\epsilon$. Poisson's Ratio for steel, for example, ranges from 0.25 to 0.3.

I. Strain Gauge

Although there are several methods of measuring strain, the most common is with a strain gauge, a device whose electrical resistance varies in proportion to the amount of strain in the device. For example, the piezoresistive strain gauge is a semiconductor device whose resistance varies nonlinearly with strain. The most widely used gauge is the bonded metallic strain gauge.

The metallic strain gauge consists of a very fine wire or, more commonly, metallic foil arranged in a grid pattern. The grid pattern maximizes the amount of metallic wire or foil subject to strain in the parallel direction as shown in the following illustration. The cross sectional area of the grid is minimized to reduce the effect of shear strain and Poisson Strain. The grid is bonded to a thin backing, called the carrier, which is attached directly to the test specimen. Therefore, the strain experienced by the test specimen is transferred directly to the strain gauge, which responds with a linear change in electrical resistance. Strain gauges are available commercially with nominal resistance values from 30 to 3,000 Ω , with 120 Ω , 350 Ω , and 1,000 Ω being the most common values.



It is very important that the strain gauge be properly mounted onto the test specimen so that the strain is accurately transferred from the test specimen through the adhesive and strain gauge backing to the foil itself. Manufacturers of strain gauges are the best source of information on proper mounting of strain gauges.

A fundamental parameter of the strain gauge is its sensitivity to strain, expressed quantitatively as the gauge factor (GF). Gauge factor is defined as the ratio of fractional change in electrical resistance to the fractional change in length (strain):

$$GF = \frac{\frac{\Delta R}{R}}{\frac{\Delta L}{L}} = \frac{\Delta R}{\epsilon}$$

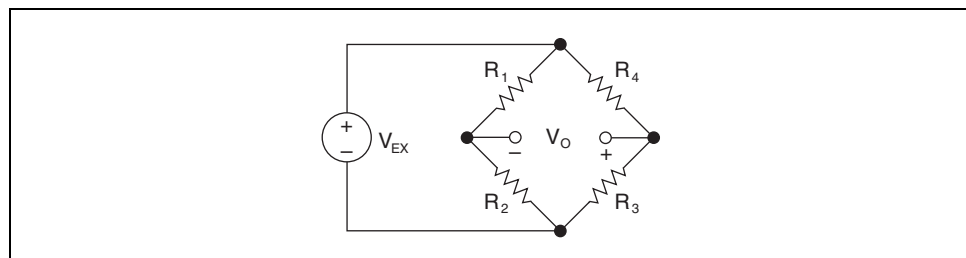
The Gauge Factor for metallic strain gauges is typically around 2.

Ideally, the resistance of the strain gauge would change only in response to applied strain. However, strain gauge material, and the specimen material to which the gauge is applied, also responds to changes in temperature. Strain gauge manufacturers attempt to minimize sensitivity to temperature by processing the gauge material to compensate for the thermal expansion of the specimen material for which the gauge is intended. While compensated gauges reduce the thermal sensitivity, they do not totally remove it. For example, consider a gauge compensated for aluminum that has a temperature coefficient of 23 ppm/°C. With a nominal resistance of 1,000 Ω, GF = 2, the equivalent strain error is still 11.5 με/°C. Therefore, additional temperature compensation is important.

Strain Gauge Measurement

Strain measurements rarely involve quantities larger than a few millistrain ($\epsilon \times 10^{-3}$). Therefore, measuring the strain requires accurate measurement of very small changes in resistance. For example, suppose a test specimen undergoes a substantial strain of 500 με. A strain gauge with a gauge factor GF = 2 exhibits a change in electrical resistance of only $2 \times (500 \times 10^{-6}) = 0.1\%$. For a 120 Ω gauge, this is a change of only 0.12 Ω.

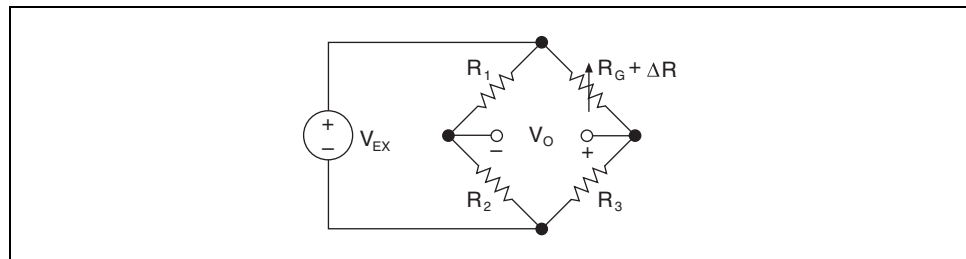
To measure such small changes in resistance and to compensate for the temperature sensitivity discussed in the previous section, strain gauges are almost always used in a bridge configuration with a voltage or current excitation source. The general Wheatstone bridge, shown in the following illustration, consists of four resistive arms with an excitation voltage, V_{EX} , that is applied across the bridge.



The output voltage of the bridge, V_O , equals

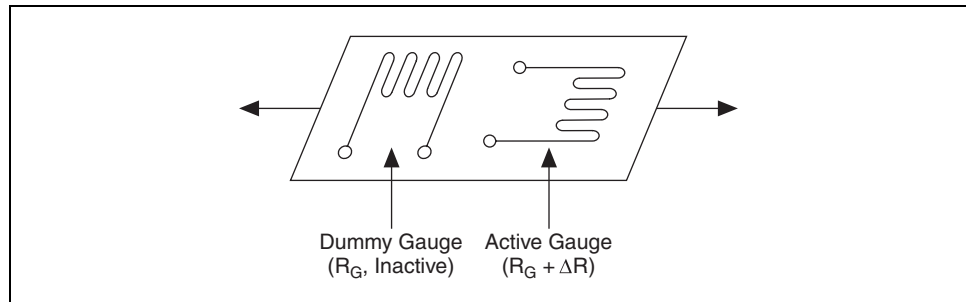
$$V_O = \left[\frac{R_3}{R_3 + R_4} - \frac{R_2}{R_1 + R_2} \right] \times V_{EX}$$

From this equation, it is apparent that when $R_1/R_2 = R_3/R_4$, the voltage output V_O is zero. Under these conditions, the bridge is said to be balanced. Any change in resistance in any arm of the bridge results in a nonzero output voltage. Therefore, if you replace R_4 in the following illustration with an active strain gauge, any changes in the strain gauge resistance unbalances the bridge and produce a nonzero output voltage. If the nominal resistance of the strain gauge is designated as R_G , the strain-induced change in resistance, ΔR , can be expressed as $\Delta R = R_G \times GF \times \epsilon$. Assuming that $R_1 = R_2$ and $R_3 = R_G$, the bridge equation above can be rewritten to express V_O/V_{EX} as a function of strain. Notice the presence of the $1/(1 + GF \times \epsilon/2)$ term that indicates the nonlinearity of the quarter-bridge output with respect to strain.

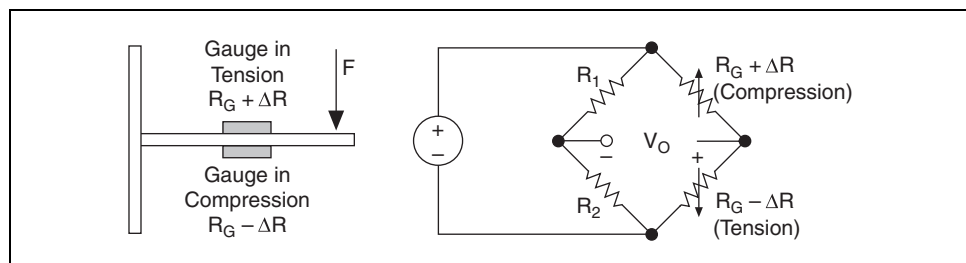


$$\frac{V_O}{V_{EX}} = -\frac{GF \times \epsilon}{4} \left(\frac{1}{1 + GF \times \frac{\epsilon}{2}} \right)$$

By using two strain gauges in the bridge, the effect of temperature can be avoided. For example, the following illustration shows a strain gauge configuration where one gauge is active ($R_G + \Delta R$), and a second gauge is placed transverse to the applied strain. Therefore, the strain has little effect on the second gauge, called the dummy gauge. However, any changes in temperature affects both gauges in the same way. Because the temperature changes are identical in the two gauges, the ratio of their resistance does not change, the voltage V_O does not change, and the effects of the temperature change are minimized.

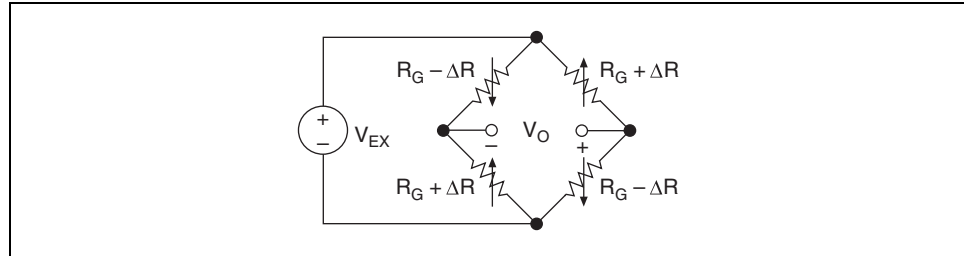


Alternatively, you can double the sensitivity of the bridge to strain by making both gauges active, although in different directions. For example, the following illustration shows a bending beam application with one bridge mounted in tension ($R_G + \Delta R$) and the other mounted in compression ($R_G - \Delta R$). This half-bridge configuration, whose circuit diagram is also shown in the following illustration, yields an output voltage that is linear and approximately doubles the output of the quarter-bridge circuit.



$$\frac{V_O}{V_{EX}} = -\frac{GF \times \epsilon}{2}$$

Finally, you can further increase the sensitivity of the circuit by making all four of the arms of the bridge active strain gauges and mounting two gauges in tension and two gauges in compression. The following illustration shows the full-bridge circuit.



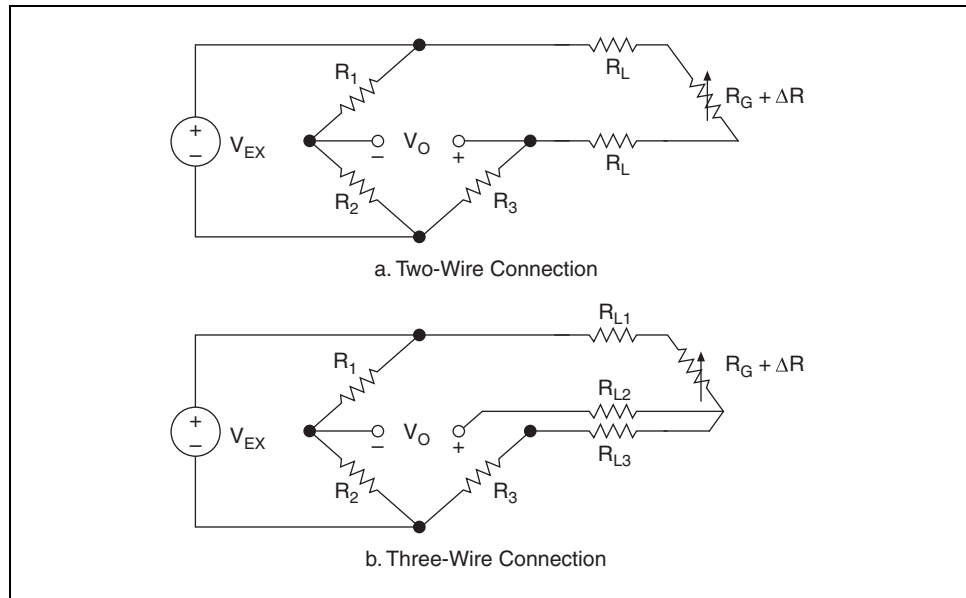
$$\frac{V_O}{V_{EX}} = -G \times \varepsilon$$

The equations given here for the Wheatstone bridge circuits assume an initially balanced bridge that generates zero output when no strain is applied. In practice, however, resistance tolerances and strain induced by gauge application generates some initial offset voltage. This initial offset voltage is typically handled in two ways. First, you can use a special offset-nulling, or balancing, circuit to adjust the resistance in the bridge to rebalance the bridge to zero output. Alternatively, you can measure the initial unstrained output of the circuit and compensate in software. Refer to the *Strain Gauge Equations* section of this lesson for equations for quarter-, half-, and full-bridge circuits that express strain and that take initial output voltages into account. These equations also include the effect of resistance in the lead wires connected to the gauges.

Lead Wire Resistance

The figures and equations in the previous section ignore the resistance in the lead wires of the strain gauge. While ignoring the lead resistances can be beneficial to understanding the basics of strain gauge measurements, doing so in practice can be very dangerous. For example, consider the two-wire connection of a strain gauge shown in the top half of the following illustration. Suppose each lead wire connected to the strain gauge is 15 m long with lead resistance R_L equal to 1 Ω . Therefore, the lead resistance adds 2 Ω of resistance to that arm of the bridge. Besides adding an offset error, the lead resistance also desensitizes the output of the bridge. From the strain equations in the *Strain Gauge Equations* section of this lesson you can see that the amount of desensitization is quantified by the term $(1 + R_L/R_G)$. You can compensate for this error by measuring the lead resistance R_L and using the measured value in the strain equations. However, a more difficult problem arises from changes in the lead resistance due to temperature changes. Given typical temperature coefficients for copper wire, a slight change in temperature can generate a measurement error of several $\mu\epsilon$.

Therefore, the preferred connection scheme for quarter-bridge strain gauges is the three-wire connection, shown in the bottom half of the following illustration. In this configuration, R_{L1} and R_{L3} appear in adjacent arms of the bridge. Any changes in resistance due to temperature cancel each other. The lead resistance in the third lead, R_{L2} , is connected to the measurement input. This lead carries very little current, and the effect of its lead resistance is negligible.

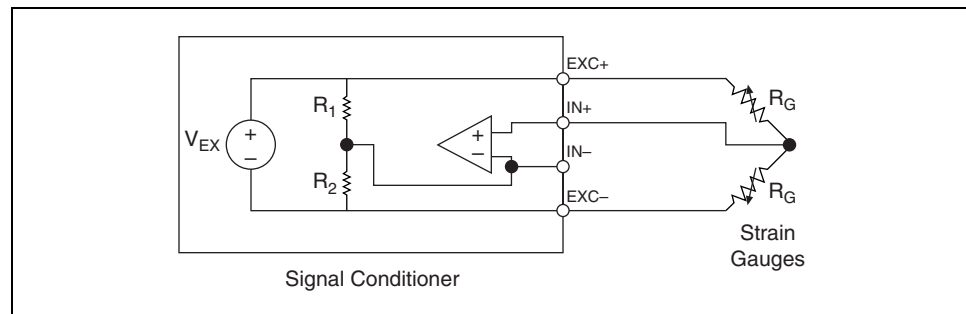


J. Signal Conditioning for Strain Gauges

Strain gauge measurement involves sensing extremely small changes in resistance. Therefore, proper selection and use of the bridge, signal conditioning, wiring, and data acquisition components are required for reliable measurements.

Bridge Completion

Unless you are using a full-bridge strain gauge sensor with four active gauges, you need to complete the bridge with reference resistors. Therefore, strain gauge signal conditioners typically provide half-bridge completion networks consisting of two high-precision reference resistors. The following illustration shows the wiring of a half-bridge strain gauge circuit to a conditioner with completion resistors R_1 and R_2 . The nominal resistance of the completion resistors is less important than how well the two resistors are matched. Ideally, the resistors are well matched and provide a stable reference voltage of $V_{EX}/2$ to the negative input lead of the measurement channel. The high resistance of the completion resistors helps minimize the current draw from the excitation voltage.



Bridge Excitation

Strain gauge signal conditioners typically provide a constant voltage source to power the bridge. While there is no standard voltage level that is recognized industry wide, excitation voltage levels of around 3 V and 10 V are common. While a higher excitation voltage generates a proportionately higher output voltage, the higher voltage also can cause larger errors due to self-heating. It is important that the excitation voltage be accurate and stable. Alternatively, you can use a less accurate or stable voltage and accurately measure, or sense, the excitation voltage so the correct strain is calculated.

Excitation Sensing

If the strain-gauge circuit is located a distance away from the signal conditioner and excitation source, a possible source of error is voltage drops caused by resistance in the wires that connects the excitation voltage to the bridge. Therefore, some signal conditioners include a feature called remote sensing to compensate for this error.

There are two common methods of remote sensing. With feedback remote sensing, you connect extra sense wires to the point where the excitation voltage wires connect to the bridge circuit. The extra sense wires serve to regulate the excitation supply to compensate for lead losses and deliver the needed voltage at the bridge.

An alternative remote sensing scheme uses a separate measurement channel to measure directly the excitation voltage delivered across the bridge. Because the measurement channel leads carry very little current, the lead resistance has negligible effect on the measurement. The measured excitation voltage is then used in the voltage-to-strain conversion to compensate for lead losses.

Signal Amplification

The output of strain gauges and bridges is relatively small. In practice, most strain-gauge bridges and strain-based transducers output less than $10 \mu\text{V}/\text{V}$ ($10 \mu\text{V}$ of output per volt of excitation voltage). With a 10 V excitation voltage, the output signal is $100 \mu\text{V}$. Therefore, strain gauge signal conditioners usually include amplifiers to boost the signal level to increase measurement resolution and improve signal-to-noise ratios. SCXI signal conditioning modules, for example, include configurable gain amplifiers with gains up to 2,000.

Bridge Balancing, Offset Nulling

When a bridge is installed, it is very unlikely that the bridge outputs exactly zero volts when no strain is applied. Rather, slight variations in resistance among the bridge arms and lead resistance generate some nonzero initial offset voltage. There are a few different ways that a system can handle this initial offset voltage.

Software Compensation

The first method compensates for the initial voltage in software. With this method, you take an initial measurement before strain input is applied. This initial voltage is then used in the strain equations listed in the *Strain Gauge Equations* section of this lesson. This method is simple, fast, and requires no manual adjustments. The disadvantage of the software compensation method is that the offset of the bridge is not removed. If the

offset is large enough, it limits the amplifier gain you can apply to the output voltage, thus limiting the dynamic range of the measurement.

Offset-Nulling Circuit

The second balancing method uses an adjustable resistance, or potentiometer, to physically adjust the output of the bridge to zero. For example, Figure 5-5 shows the offset-nulling circuit of the SCXI-1321 terminal block. By varying the position of the potentiometer (R_{POT}), you can control the level of the bridge output and set the initial output to zero volts. The value of R_{NULL} sets the range that the circuit can balance.

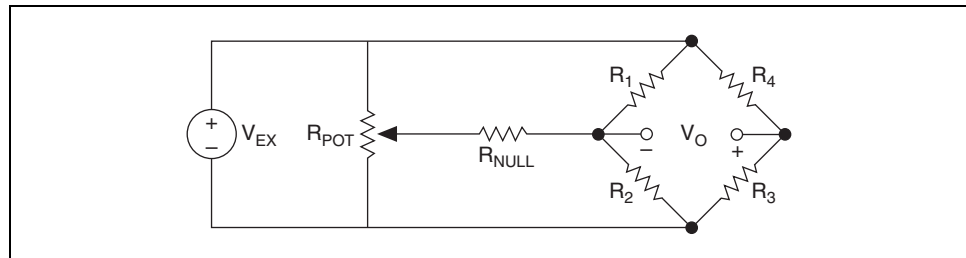


Figure 5-5. Offset-Nulling Circuit of SCXI-1321 Terminal Block

Buffered Offset Nulling

The third method, like the software method, does not affect the bridge directly. With buffered nulling, a nulling circuit adds an adjustable DC voltage to the output of the instrumentation amplifier. For example, the SC-2043-SG strain gauge accessory uses this method. The SC-2043-SG includes a user-adjustable potentiometer that can add $\pm 50 \mu\text{V}$ to the output of an instrumentation amplifier that has a fixed gain of 10. Therefore, the nulling range, referred to input, is $\pm 5 \mu\text{V}$.

Shunt Calibration

The normal procedure to verify the output of a strain-gauge measurement system relative to some predetermined mechanical input or strain is called shunt calibration. Shunt calibration involves simulating the input of strain by changing the resistance of an arm in the bridge by some known amount. This is accomplished by shunting, or connecting, a large resistor of known value across one arm of the bridge, creating a known ΔR . You can measure the output of the bridge and compare it to the expected voltage value. You can use the results to correct span errors in the entire measurement path or to simply verify general operation to gain confidence in the setup.

K. Strain Gauge Equations

This section includes the complete strain-gauge equations for several types of bridge configurations. These equations are included as callable functions (with source code) in NI-DAQ . The function names are `Strain_Convert` and `Strain_Buf_Convert`. In LabVIEW, these equations are included in the Convert Strain Gauge Reading VI located on the **Functions»NI Measurements»Data Acquisition»Signal Conditioning** palette.

To simplify the equations and account for unbalanced bridges in the nonstrained state, the ratio V_r is

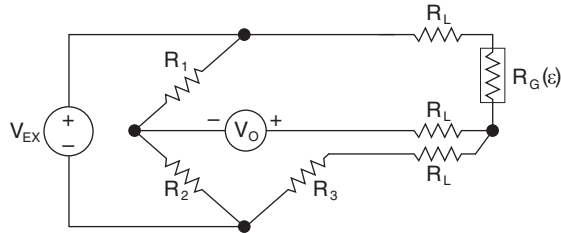
$$V_r = \frac{V_{O(\text{strained})} - V_{O(\text{unstrained})}}{V_{EX}}$$

where $V_{O(\text{strained})}$ is the measured output when strained, and $V_{O(\text{unstrained})}$ is the initial, unstrained output voltage. V_{EX} is the excitation voltage.

Also, the designation (+ ϵ) and (- ϵ) indicates active strain gauges mounted in tension and compression, respectively. The designation (- $\nu\epsilon$) indicates that the strain gauge is mounted in the transversal direction, so that its resistance change is primarily due to the Poisson's Strain, whose magnitude is given as - $\nu\epsilon$.

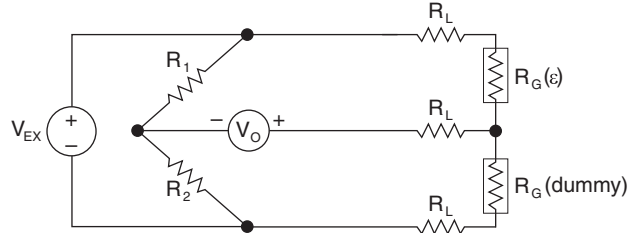
Other nomenclature used in the equations include:

- R_G = nominal resistance value of strain gauge
- GF = gauge factor of strain gauge
- R_L = lead resistance



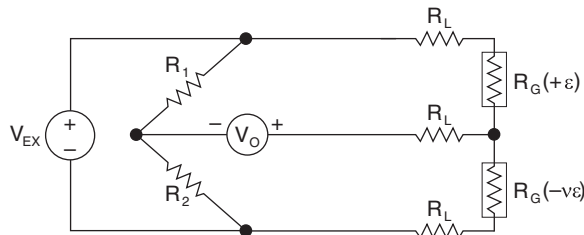
Quarter-Bridge I

$$strain(\epsilon) = \frac{-4V_r}{GF(1 + 2V_r)} \cdot \left(1 + \frac{R_L}{R_G}\right)$$



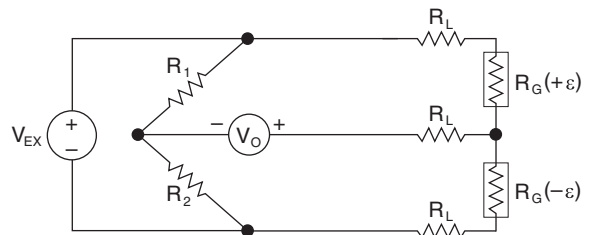
Quarter-Bridge II

$$strain(\epsilon) = \frac{-4V_r}{GF(1 + 2V_r)} \cdot \left(1 + \frac{R_L}{R_G}\right)$$



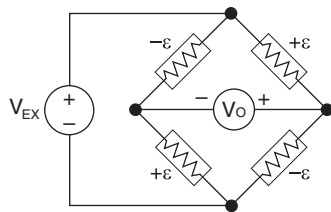
Half-Bridge I

$$strain(\epsilon) = \frac{-4V_r}{GF[(1 + v) - 2V_r(v - 1)]} \cdot \left(1 + \frac{R_L}{R_G}\right)$$



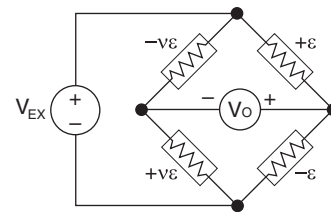
Half-Bridge II

$$strain(\epsilon) = \frac{-2V_r}{GF} \cdot \left(1 + \frac{R_L}{R_G}\right)$$



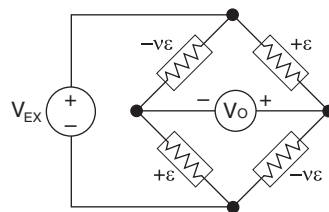
Full-Bridge I

$$strain(\epsilon) = \frac{-V_r}{GF}$$



Full-Bridge II

$$strain(\epsilon) = \frac{-2V_r}{GF(v + 1)}$$



Full-Bridge III

$$strain(\epsilon) = \frac{-2V_r}{GF[(v + 1) - V_r(v - 1)]}$$

Exercise 5-5 Strain Reading

Objective: To make a strain gauge measurement.

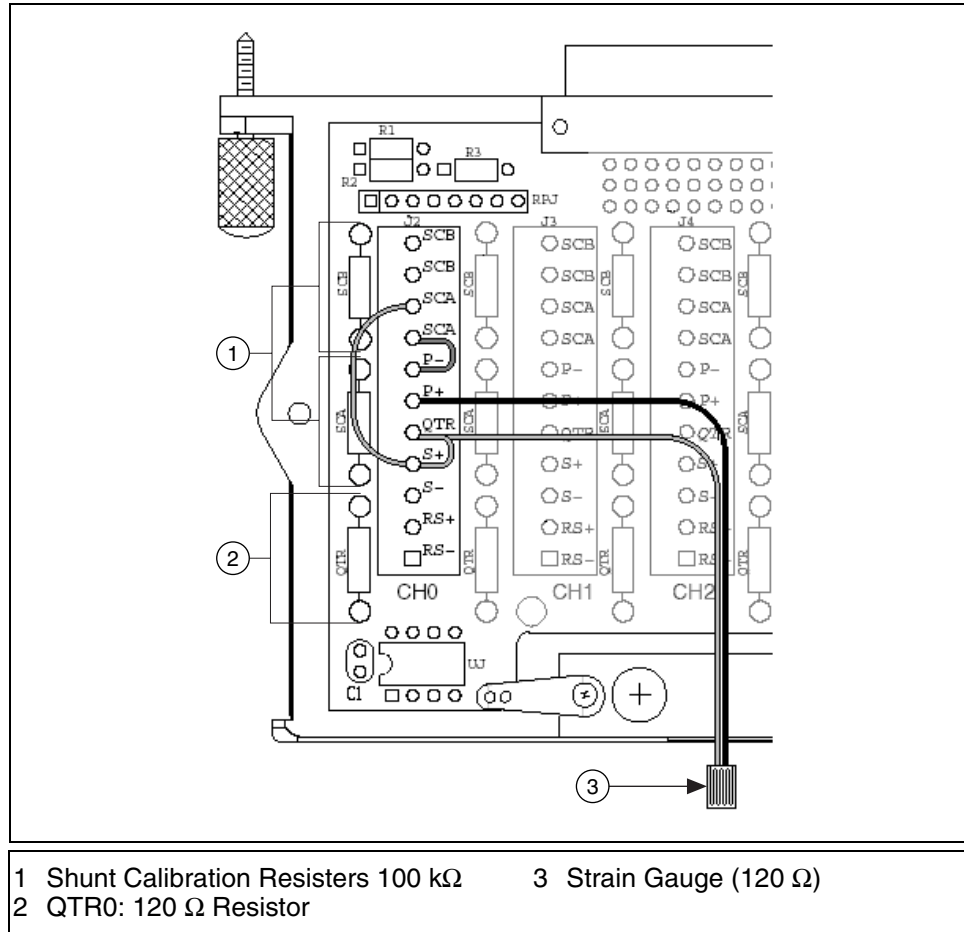
In this exercise, you will acquire data from a strain gauge connected to the SCXI-1520 and programmatically apply shunt calibration and offset nulling to increase the accuracy of the reading.

Connect the Strain Gauge to CH 0 of the SCXI-1314 Terminal Block



Caution SCXI modules are *not* hot-swappable. Make sure the chassis is powered off before removing or adding any modules, terminal blocks, or cables.

1. Power off the SCXI chassis.
2. Remove the cover from the SCXI-1314 terminal block.
3. Look at the SCXI-1314 and find QTR0, as shown in the following illustration.



This is the quarter-bridge I completion resistor. This exercise uses a 120 Ω strain gauge that requires quarter-bridge completion. The quarter-bridge completion resistors for the channels are socketed so you can change the resistors as needed. All SCXI-1314 universal strain gauge terminal blocks for this course have the 120 Ω resistor installed.

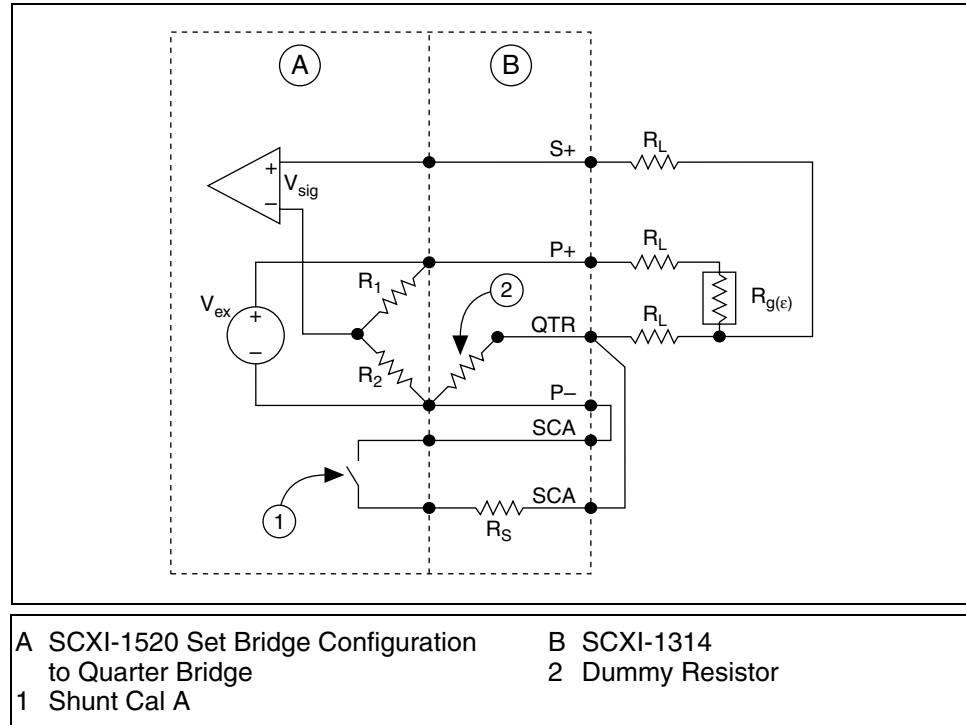


Note The 120 Ω resistor must be a precision resistor. Check with your instructor to verify that the resistor is a precision resistor.

4. Wire the strain gauge to CH0 of the SCXI-1314 using the previous illustration and the following steps.
 - a. The terminal assignments are as follows:
 - S+ and S– are for analog input.
 - RS+ and RS– are for remote sense (not used in this exercise).
 - P+ and P– are for excitation output.
 - SCA are for shunt calibration circuit A.
 - SCB are for shunt calibration circuit B (not used in this exercise).

- b. Connect one wire of the strain gauge to P+ of CH0.
- c. Connect the other wire of the strain gauge to QTR of CH0.
- d. Connect a wire from QTR to S+.
- e. Connect a wire from S+ to one of the two SCA terminals.
- f. Connect a wire from P- to the other SCA terminal.

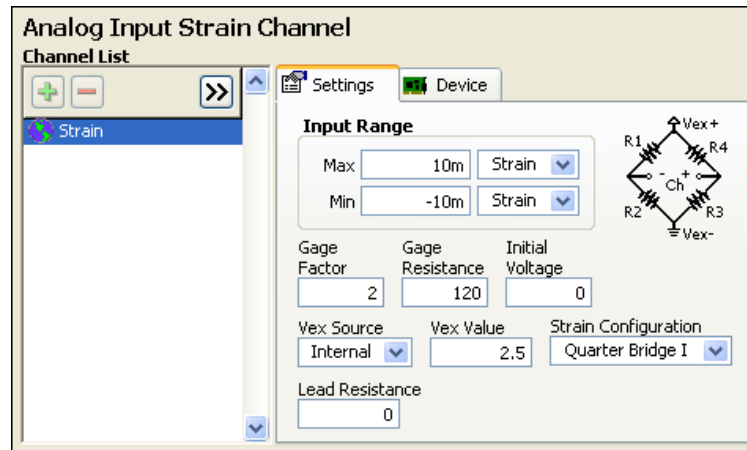
The following illustration shows the schematic for quarter-bridge I completion.



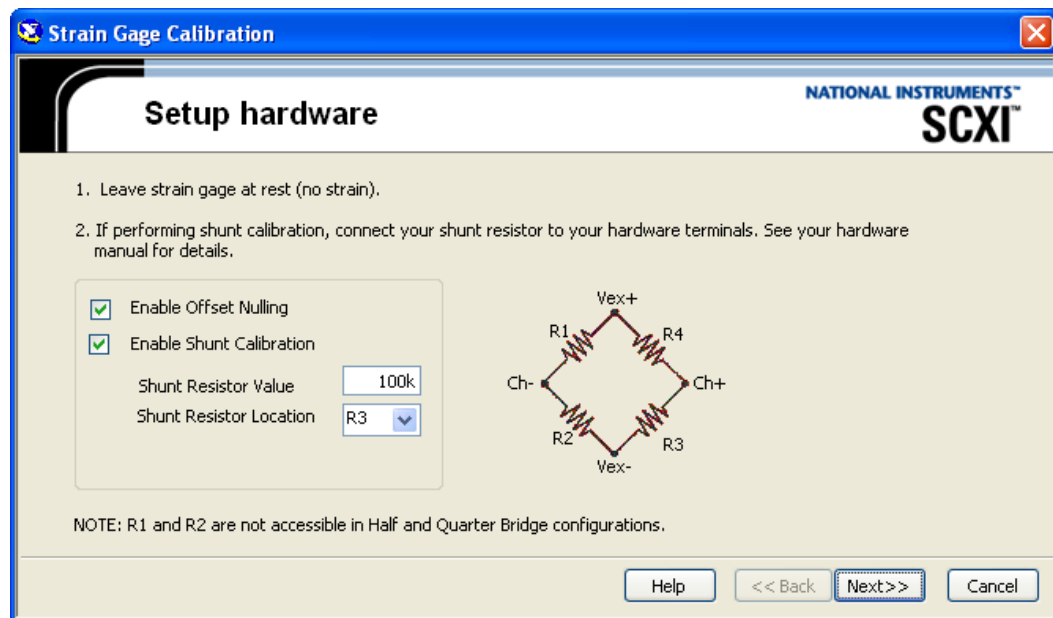
5. Replace the SCXI-1314 cover.
6. Attach the SCXI-1314 to the SCXI-1520.
7. Power on the SCXI chassis.

Calibration in MAX

1. Launch MAX and create a new NI-DAQmx global channel to read a strain input (Analog Input Strain Channel). Name the channel `Strain`.
2. In the DAQ Assistant, use the following settings.



3. Click the **Device** tab. Leave the settings at the default values and click **Calibration** to open the **Strain Gage Calibration** dialog box.



4. Click the **Next** button.
5. The strain gauge will be automatically measured. Notice the **Err %** columns for the **Offset Adjustment** and the **Gain Adjustment (with shunt)**.
6. Click the **Calibrate** button. The error values should be very close to zero.
7. Click the **Finish** button.
8. Close MAX. Do not save the Strain channel.

Strain Readings with Programmatic Shunt Calibration and Offset Nulling

1. Launch LabVIEW and select **Help»Find Examples** to open the NI Example Finder. Browse by task to **Hardware Input and Output»DAQmx»Analog Measurements»Strain** and open the Acq Strain Samples (with Calibration) VI.
2. Examine the block diagram. Notice the use of the DAQmx Channel Property Node.
3. Switch to the front panel and set the front panel controls with the following values.
 - **Physical Channels:** SC1Mod1/ai0
 - **Input Limits: High:** 0.01, **Low:** -0.01
 - **Filter Enabled?:** TRUE (Right-click the control and select **Data Operations»Change Value to True** from the shortcut menu.)
 - **Gauge Factor:** Ask the instructor for this value
 - **Nominal Gage Resistance:** 120
 - **Strain Configuration:** Quarter Bridge I (Select this value from the pull-down menu, do not type in the name.)
 - **Excitation Voltage:** 2.500
 - **Do Strain Null?:** TRUE
 - **Do Shunt Cal?:** TRUE
 - **Shunt Location:** R3
 - **Shunt Resistance:** 100000
 - **Measure Actual Excitation?:** TRUE
4. Run the VI.
5. A few seconds will pass as the VI performs the strain nullification and shunt calibration.
6. Apply compression and tension to the strain gauge. You should notice the meter reading fluctuate above or below zero as you apply strain to the metal bar attached to the strain gauge.
7. Stop and close the VI. Do not save changes.

End of Exercise 5-5

Exercise 5-6 SCXI Disassembly

Objective: To remove all the transducers and terminal blocks from the SCXI chassis.



Caution SCXI modules are *not* hot-swappable. Make sure the chassis is powered off before removing or adding any modules, terminal blocks, or cables.

1. Power off the SCXI chassis.
2. Remove the SCXI-1304, SCXI-1327, SCXI-1314, and SCXI-1302 terminal blocks from the chassis.
3. Remove the top covers on the terminal blocks and remove all jumper wires and transducers.
4. Replace the top covers to the terminal blocks.
5. Gently remove the 68-pin cable on the back of the chassis and reconnect the cable to the DAQ Signal Accessory.
6. Launch MAX. Double-click **Devices and Interfaces»NI-DAQmx Devices**.
7. Delete the **SCXI-1000** device and exit MAX.

End of Exercise 5-6

Summary

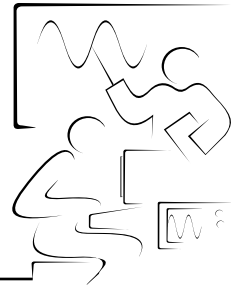
- Signal conditioning can be accomplished using SCXI or SCC.
- Signal conditioning equipment is controlled by the DAQ device.
- Signal conditioning provides amplification, multiplexing, filtering, isolation, and so on.
- Signal conditioning provides the necessary filtering, isolation, and amplification for thermocouples.
- Strain gauge measurements can be easily accomplished using signal conditioning.

Notes

Notes

Lesson 6

Signal Processing



This lesson describes the basics of signal processing.

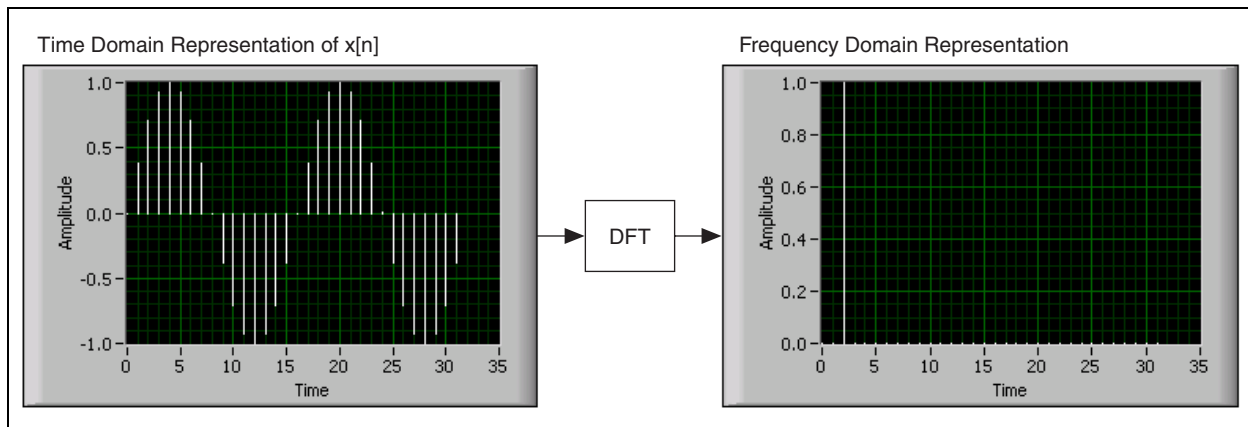
You Will Learn:

- A. About the discrete Fourier transform (DFT) and the fast Fourier transform (FFT)
- B. About magnitude and phase
- C. About spectral leakage and smoothing windows
- D. About filtering and why it is needed
- E. Differences between IIR and FIR filters

A. Discrete Fourier Transform (DFT) and Fast Fourier Transform (FFT)

The samples of a signal obtained from a DAQ device constitute the time-domain representation of the signal. This representation gives the amplitudes of the signal at the instant of time during which it is sampled. In many cases you might want to know the frequency content of a signal rather than the amplitudes of the individual samples. The representation of a signal in terms of its individual frequency components is known as the frequency-domain representation of the signal. The frequency-domain representation can give more insight about the signal and the system from which it was generated.

The algorithm used to transform samples of the data from the time domain into the frequency domain is known as the discrete Fourier transform or DFT. The DFT establishes the relationship between the samples of a signal in the time domain and their representation in the frequency domain, as shown in the following figure. The DFT is used in the fields of spectral analysis, applied mechanics, acoustics, medical imaging, numerical analysis, instrumentation, and telecommunications.



If you obtain N samples of a signal from a DAQ device and apply the DFT to N samples of the time-domain representation of the signal, the result is also of length N samples, but the information it contains is of the frequency-domain representation.

If the signal is sampled at a sampling rate of f_s Hz, the time interval between the samples (the sampling interval) is Δt , where

$$\Delta t = \frac{1}{f_s}$$

The sample signals are denoted by $x[i]$, $0 \leq i \leq N - 1$ (that is, you have a total of N samples). When the discrete Fourier transform, given by

$$X_k = \sum_{i=0}^{N-1} x_i e^{-j2\pi i k / N} \quad \text{for } k = 0, 1, 2, \dots, N-1 \quad (6-1)$$

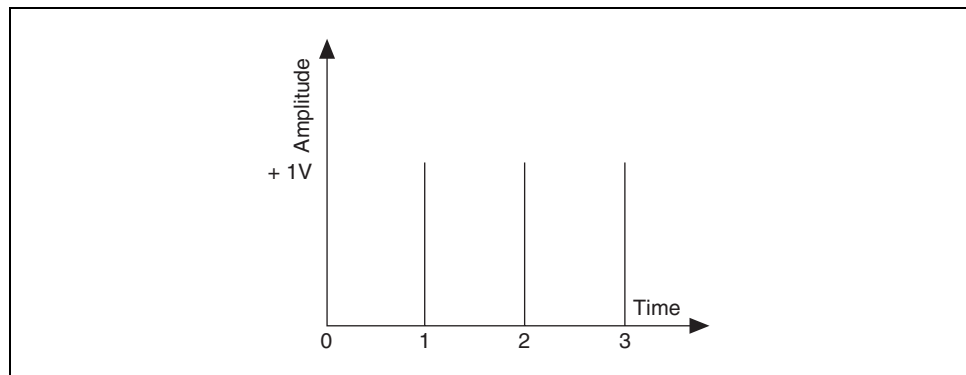
is applied to these N samples, the resulting output ($X[k]$, $0 \leq k \leq N - 1$) is the frequency-domain representation of $x[i]$. Notice that both the time-domain x and the frequency-domain X have a total of N samples. Analogous to the time spacing of Δt between the samples of x in the time domain, you have a frequency spacing of

$$\Delta f = \frac{f_s}{N} = \frac{1}{N\Delta t}$$

between the components of X in the frequency domain. Δf also is known as the frequency resolution. To increase the frequency resolution (smaller Δf), either increase the number of samples N with f_s constant or decrease the sampling frequency f_s with N constant.

DFT Calculation Example

Assume that $X[0]$ corresponds to DC, or the average value, of the signal. To see the result of calculating the DFT of a waveform with the use of Equation 6-1, consider a DC signal having a constant amplitude of +1 V. Four samples of this signal are taken, as shown in the following illustration.



Each of the samples has a value +1, giving the time sequence

$$x[0] = x[1] = x[2] = x[3] = 1$$

Using Equation 6-1 to calculate the DFT of this sequence and making use of Euler's identity,

$$\exp(-j\theta) = \cos(\theta) - j\sin(\theta)$$

results in:

$$X[0] = \sum_{i=0}^{N-1} x_i e^{-j2\pi i0/N} = x[0] + x[1] + x[2] + x[3] = 4$$

$$X[1] = x[0] + x[1]\left(\cos\left(\frac{\pi}{2}\right) - j\sin\left(\frac{\pi}{2}\right)\right) + x[2](\cos(\pi) - j\sin(\pi)) + x[3]\left(\cos\left(\frac{3\pi}{2}\right) - j\sin\left(\frac{3\pi}{2}\right)\right) = (1 - j - 1 + j) = 0$$

$$X[2] = x[0] + x[1](\cos(\pi) - j\sin(\pi)) + x[2](\cos(2\pi) - j\sin(2\pi)) + x[3](\cos(3\pi) - j\sin(3\pi)) = (1 - 1 + 1 - 1) = 0$$

$$X[3] = x[0] + x[1]\left(\cos\left(\frac{3\pi}{2}\right) - j\sin\left(\frac{3\pi}{2}\right)\right) + x[2](\cos(3\pi) - j\sin(3\pi)) + x[3]\left(\cos\left(\frac{9\pi}{2}\right) - j\sin\left(\frac{9\pi}{2}\right)\right) = (1 + j - 1 - j) = 0$$

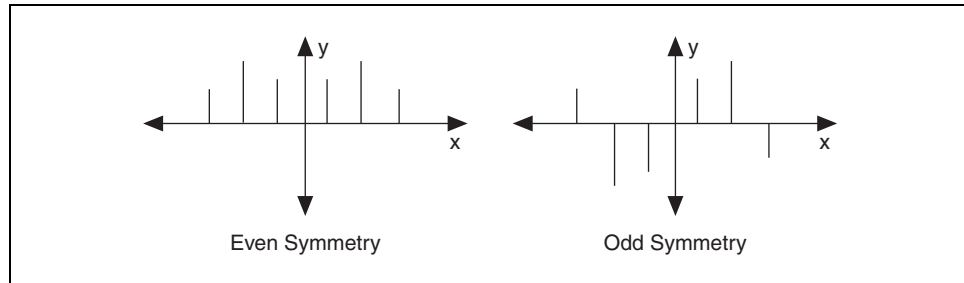
Except for the DC component, $X[0]$, all the other values are zero, as expected. However, the calculated value of $X[0]$ depends on the value of N (the number of samples). Because you had $N = 4$, $X[0] = 4$. If $N = 10$, you would have calculated $X[0] = 10$. This dependency of $X[.]$ on N also occurs for the other frequency components. Usually, you divide the DFT output by N to obtain the correct magnitude of the frequency component.

Magnitude and Phase Information

You have seen that N samples of the input signal result in N samples of the DFT. That is, the number of samples in both the time and frequency representations is the same. From Equation 6-1, you see that regardless of whether the input signal $x[i]$ is real or complex, $X[k]$ is always complex, although the imaginary part may be zero. Because the DFT is complex, it contains two pieces of information: the amplitude and the phase. For real signals ($x[i]$ real), such as those obtained from the output of one channel of a DAQ device, the DFT is symmetric about the index $N/2$ with the following properties:

$$|X[k]| = |X[N-k]| \text{ and phase}(X[k]) = -\text{phase}(X[N-k])$$

The magnitude of $X[k]$ is known as even symmetric, and $\text{phase}(X[k])$ is known as odd symmetric. An even symmetric signal is one that is symmetric about the y-axis, whereas an odd symmetric signal is symmetric about the origin. The following illustration demonstrates this symmetry.



The net effect of this symmetry is repetition of information contained in the N samples of the DFT. Because of this repetition of information, only half of the samples of the DFT actually need to be computed or displayed. The other half can be obtained from this repetition.



Note If the input signal is complex, the DFT is nonsymmetrical, and you cannot use this method.

B. Frequency Spacing and Symmetry of the DFT/FFT

Because the sampling interval is Δt seconds, and if the first ($k = 0$) data sample is assumed to be at 0 seconds, the k^{th} ($k > 0$, k integer) data sample is at $k\Delta t$ seconds. Similarly, the frequency resolution being Δf , where ($\Delta f = \frac{f_s}{N}$), means that the k^{th} sample of the DFT occurs at a frequency of $k\Delta f$ Hz. This is valid for only up to about half the number of samples. The other half represent negative frequency components. Depending on if the number of samples, N , is even or odd, you can have a different interpretation of the frequency that corresponds to the k^{th} sample of the DFT.

Even Number of Samples

Suppose N is even and let $p = \frac{N}{2}$.

The following table shows the frequency to which each element of the complex output sequence X corresponds.

Array Element	Corresponding Frequency
$X[0]$	DC component
$X[1]$	Δf
$X[2]$	$2\Delta f$
$X[3]$	$3\Delta f$
\vdots	\vdots
$X[p-2]$	$(p-2)\Delta f$
$X[p-1]$	$(p-1)\Delta f$
$X[p]$	$p\Delta f$ (Nyquist frequency)
$X[p+1]$	$-(p-1)\Delta f$
$X[p+2]$	$-(p-2)\Delta f$
\vdots	\vdots
$X[N-3]$	$-3\Delta f$

Array Element	Corresponding Frequency
$X[N-2]$	$-2\Delta f$
$X[N-1]$	$-1\Delta f$

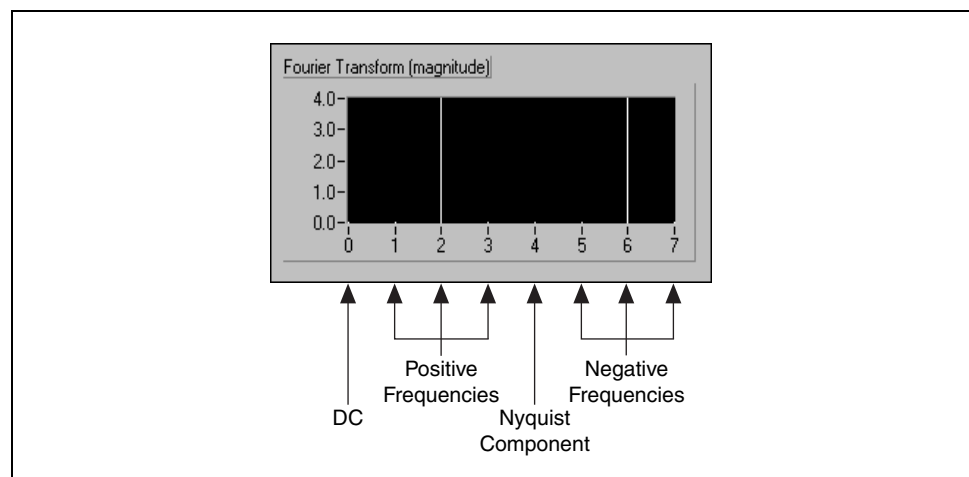
The p^{th} element, $X[p]$, corresponds to the Nyquist frequency. The negative entries in the second column beyond the Nyquist frequency represent negative frequencies.

For example, if $N = 8$, $p = N/2 = 4$, then

$X[0]$ DC
 $X[1]$ Δf
 $X[2]$ $2\Delta f$
 $X[3]$ $3\Delta f$
 $X[4]$ $4\Delta f$ (Nyquist freq)
 $X[5]$ $-3\Delta f$
 $X[6]$ $-2\Delta f$
 $X[7]$ $-\Delta f$

$X[1]$ and $X[7]$ have the same magnitude, $X[2]$ and $X[6]$ have the same magnitude, and $X[3]$ and $X[5]$ have the same magnitude. The difference is that whereas $X[1]$, $X[2]$, and $X[3]$ correspond to positive frequency components, $X[5]$, $X[6]$, and $X[7]$ correspond to negative frequency components. Notice that $X[4]$ is at the Nyquist frequency.

The following illustration represents this complex sequence for $N = 8$.



This type of representation, where you see both the positive and negative frequencies, is known as the two-sided transform.

Odd Number of Samples

Suppose that N is odd. Let $p = (N - 1)/2$. The following table shows the frequency to which each element of the complex output sequence X corresponds.

Array Element	Corresponding Frequency
$X[0]$	DC component
$X[1]$	Δf
$X[2]$	$2\Delta f$
$X[3]$	$3\Delta f$
\vdots	\vdots
$X[p-1]$	$(p-1)\Delta f$
$X[p]$	$p\Delta f$
$X[p+1]$	$-p\Delta f$
$X[p+2]$	$-(p-1)\Delta f$
\vdots	\vdots
$X[N-3]$	$-3\Delta f$
$X[N-2]$	$-2\Delta f$
$X[N-1]$	$-\Delta f$

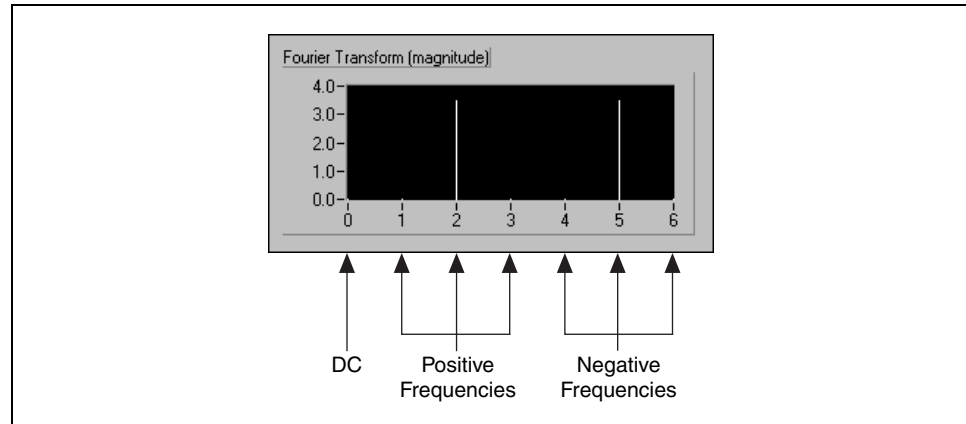
When N is odd, $N/2$ is not an integer. Therefore, there is no component at the Nyquist frequency.

If $N = 7$, $p = (N-1)/2 = (7-1)/2 = 3$, and you have

$X[0]$ DC
 $X[1]$ Δf
 $X[2]$ $2\Delta f$
 $X[3]$ $3\Delta f$
 $X[4]$ $-3\Delta f$
 $X[5]$ $-2\Delta f$
 $X[6]$ $-\Delta f$

$X[1]$ and $X[6]$ have the same magnitude, $X[2]$ and $X[5]$ have the same magnitude, and $X[3]$ and $X[4]$ have the same magnitude. However, whereas $X[1]$, $X[2]$, and $X[3]$ correspond to positive frequencies, $X[4]$, $X[5]$, and $X[6]$ correspond to negative frequencies. Because N is odd, there is no component at the Nyquist frequency.

The following illustration represents the preceding table for $N = 7$.



This is also a two-sided transform because you have both positive and negative frequencies.

Fast Fourier Transforms

Direct implementation of the DFT, as in Equation 6-1, on N data samples requires approximately N^2 complex operations and is a time-consuming process. However, when the size of the sequence is a power of 2,

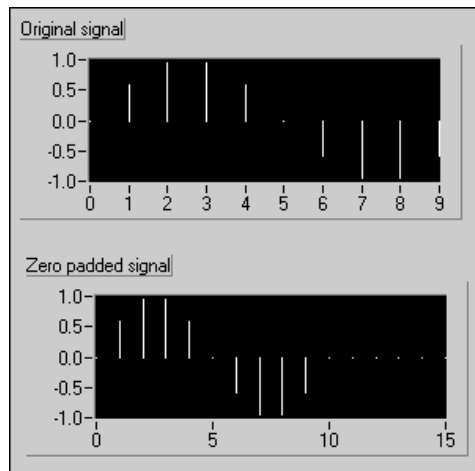
$$N = 2^m \text{ for } m = 1, 2, 3, \dots$$

you can implement the computation of the DFT with approximately $N \log_2(N)$ operations. This makes the calculation of the DFT much faster. DSP literature refers to these algorithms as fast Fourier transforms (FFTs). The FFT is a fast algorithm for calculating the DFT when the number of samples (N) is a power of 2.

The advantages of the FFT include speed and memory efficiency because the VI performs the transform in place. The size of the input sequence must be a power of 2. The DFT can process any size sequence efficiently, but the DFT is slower than the FFT and uses more memory because it stores intermediate results during processing.

Zero Padding

A technique employed to make the input sequence size equal to a power of 2 is to add zeros to the end of the sequence so that the total number of samples is equal to the next higher power of 2. For example, if you have 10 samples of a signal, you can add six zeros to make the total number of samples equal to 16 ($= 2^4$, a power of 2). The following figure illustrates this concept.



In addition to making the total number of samples a power of 2 so that faster computation is made possible by using the FFT, zero padding also helps increase the frequency resolution (recall that $\Delta f = f_s/N$) by increasing the number of samples, N .

C. Power Spectrum

The DFT or FFT of a real signal is a complex number, having real and imaginary parts. The power in each frequency component represented by the DFT/FFT can be obtained by squaring the magnitude of that frequency component. Therefore, the power in the k^{th} frequency component (the k^{th} element of the DFT/FFT) is given by $|X[k]|^2$. The plot showing the power in each of the frequency components is known as the power spectrum. Because the DFT/FFT of a real signal is symmetric, the power at a positive frequency of $k\Delta f$ is the same as the power at the corresponding negative frequency of $-k\Delta f$, DC and Nyquist components not included. The total power in the DC and Nyquist components consists of

$$|X[0]|^2 \text{ and } \left|X\left[\frac{N}{2}\right]\right|^2, \text{ respectively.}$$

Loss of Phase Information

Because power is obtained by squaring the magnitude of the DFT/FFT, the power spectrum is always real, and all the phase information is lost. To obtain phase information, use the DFT/FFT, which gives you a complex output.

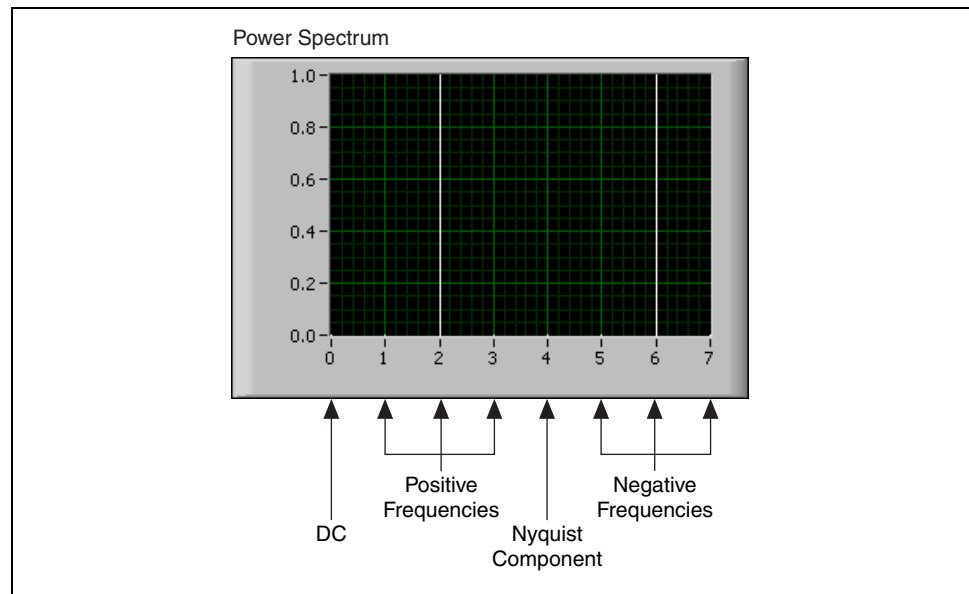
You can use the power spectrum in applications where phase information is not necessary; for example, to calculate the harmonic power in a signal. You can apply a sinusoidal input to a nonlinear system and see the power in the harmonics at the system output.

Frequency Spacing between Samples

The frequency spacing between the output samples is $\Delta f = fs/N$. In the following table, the power spectrum of a signal $x[n]$ is represented by S_{xx} . If N is even, let $p = \frac{N}{2}$. The table shows the format of the output sequence S_{xx} corresponding to the power spectrum.

Array Element	Interpretation
$S_{xx}[0]$	Power in DC component
$S_{xx}[1] = S_{xx}[N-1]$	Power at frequency Δf
$S_{xx}[2] = S_{xx}[N-2]$	Power at frequency $2\Delta f$
$S_{xx}[3] = S_{xx}[N-3]$	Power at frequency $3\Delta f$
⋮	⋮
$S_{xx}[p-2] = S_{xx}[N-(p-2)]$	Power at frequency $(p-2)\Delta f$
$S_{xx}[p-1] = S_{xx}[N-(p-1)]$	Power at frequency $(p-1)\Delta f$
$S_{xx}[p]$	Power at Nyquist frequency

The following illustration represents the information in the previous table for a sine wave with amplitude = $2 V_{\text{peak}} (V_{\text{pk}})$, and $N = 8$.



The output units of the power spectrum calculation are in volts rms squared (V_{rms}^2). So, if the peak amplitude (V_{pk}) of the input signal is $2 V_{\text{pk}}$, its

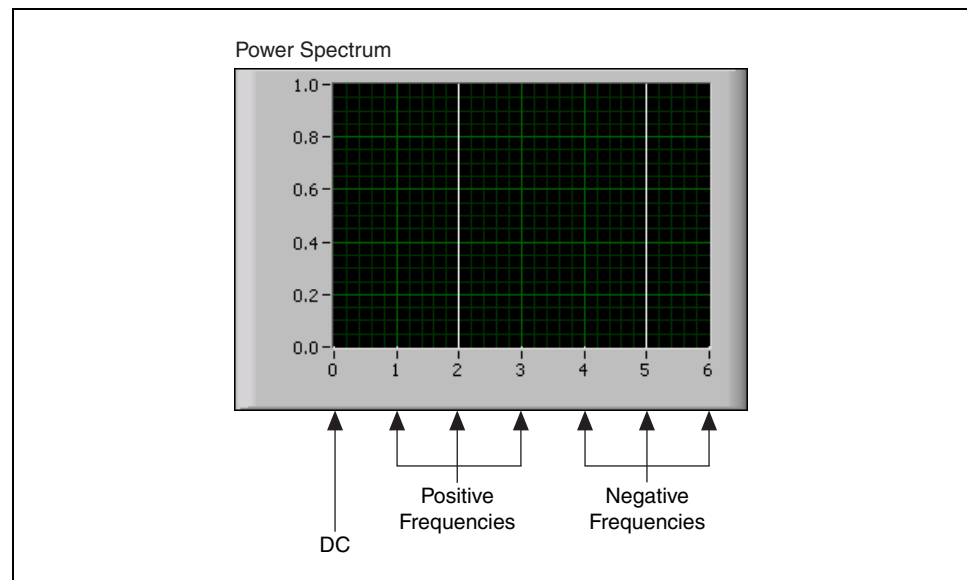
rms value is $V_{\text{rms}} = \frac{2}{\sqrt{2}} = \sqrt{2}$, so $V_{\text{rms}}^2 = 2$. This value is divided equally

between the positive and negative frequency components, resulting in the plot shown in the previous illustration.

If N is odd, let $p = (N - 1)/2$. The following table shows the format of the output sequence S_{xx} corresponding to the power spectrum.

Array Element	Interpretation
$S_{xx}[0]$	Power in DC component
$S_{xx}[1] = S_{xx}[N-1]$	Power at frequency Δf
$S_{xx}[2] = S_{xx}[N-2]$	Power at frequency $2\Delta f$
$S_{xx}[3] = S_{xx}[N-3]$	Power at frequency $3\Delta f$
⋮	⋮
$S_{xx}[p-2] = S_{xx}[N-(p-2)]$	Power at frequency $(p-2)\Delta f$
$S_{xx}[p-1] = S_{xx}[N-(p-1)]$	Power at frequency $(p-1)\Delta f$
$S_{xx}[p] = S_{xx}[p]$	Power at frequency $p\Delta f$

The following illustration represents the information in the preceding table for $N = 7$.



Spectral Measurements Express VI

The Spectral Measurements Express VI, located on the Signal Analysis palette, allows you to easily perform various spectral measurements on a signal. These measurements include computing the power spectrum and peak magnitude of a signal. Use the Configure Spectral Measurements property page to configure this Express VI to perform either a magnitude (peak), magnitude (RMS), power spectrum, or power spectral density computation in either linear or dB mode. Additionally, you can specify a windowing mode, averaging parameters, and phase of the spectral measurement. Refer to the *About Spectral Leakage and Smoothing Windows* section of this lesson for more information about windowing modes.

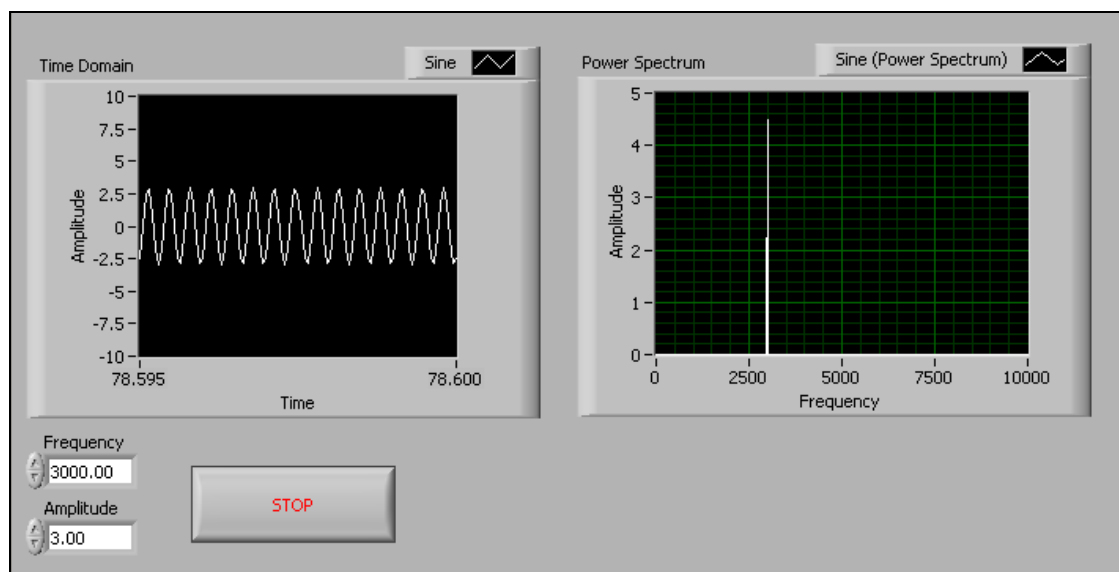
Exercise 6-1 Power Spectrum

Objective: To build a VI that determines the power spectrum of a generated signal.

The FFT is a powerful computational method you can use to perform a spectral analysis on discrete sampled data such as that from a DAQ system. This exercise uses LabVIEW Express VIs to generate a signal and perform spectral analysis of the data.

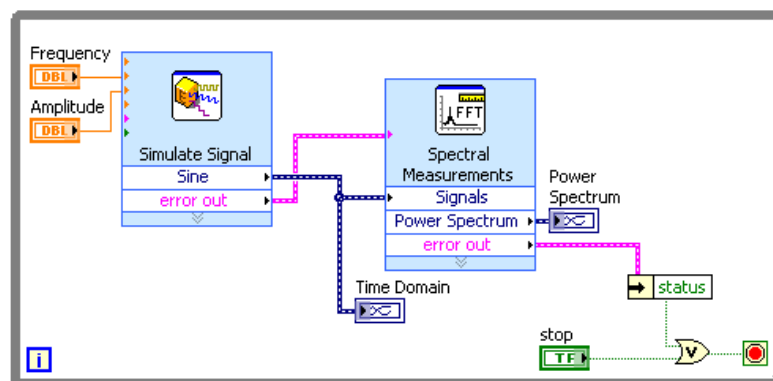
Front Panel

1. Open a blank VI.
2. The following front panel will result from building the block diagram.



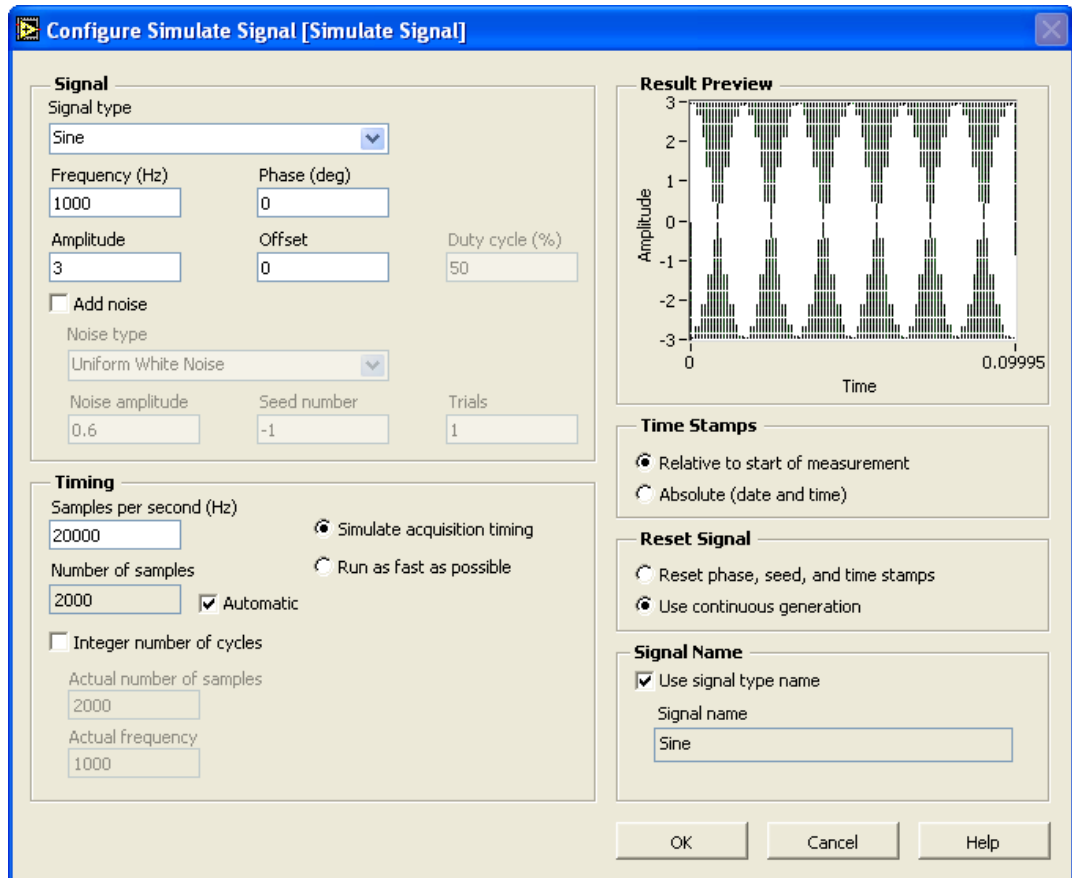
Block Diagram

3. Build the following block diagram.





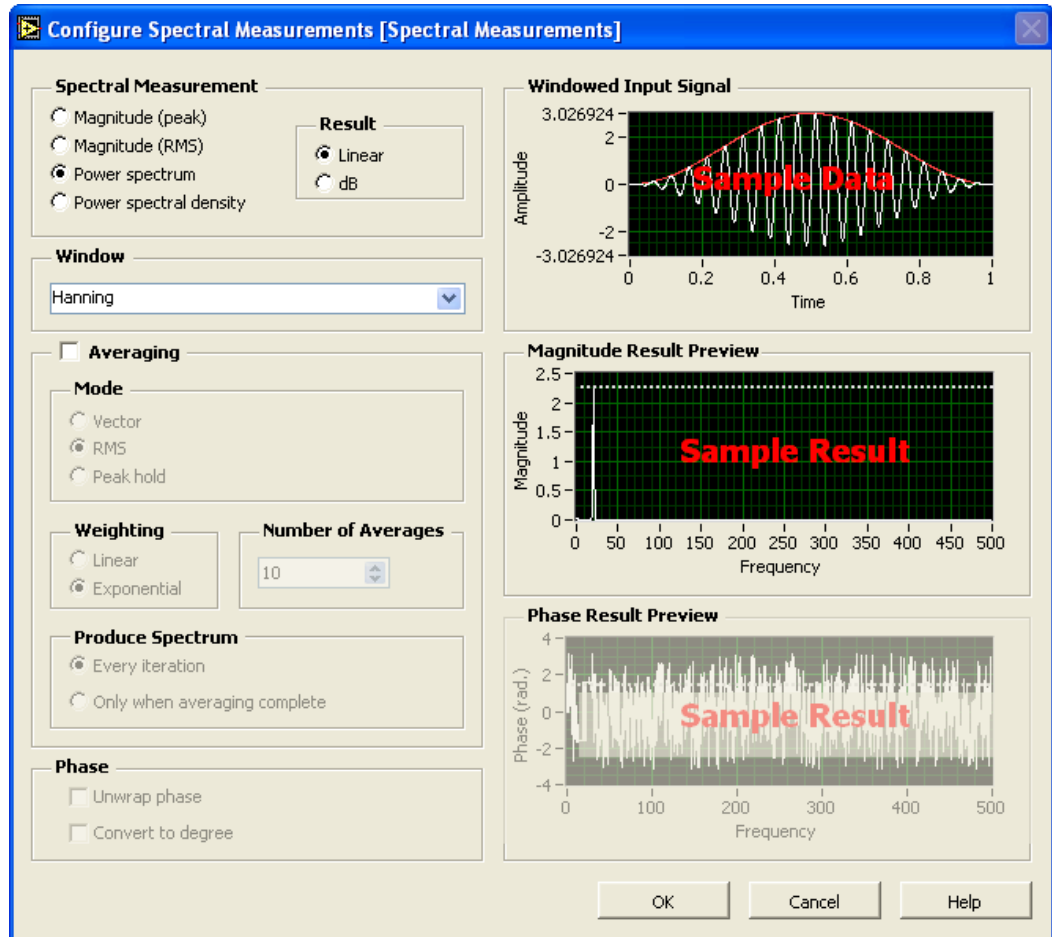
- a. Place a Simulate Signal Express VI, located on the **Functions» Signal Analysis** palette, on the block diagram. This Express VI creates an output signal of a specified type. In the **Configure Simulate Signals** dialog box that appears, select the settings shown in the following figure.



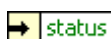
- (1) Click the **OK** button to close the dialog box.
- (2) Resize the Simulate Signals Express VI to display the **error out** output.
- (3) Right-click the **Frequency** input and select **Create»Control** from the shortcut menu.
- (4) Right-click the **Amplitude** input and select **Create»Control** from the shortcut menu.



- b. Place the Spectral Measurements Express VI, located on the **Functions»Signal Analysis** palette, on the block diagram. This Express VI computes the power spectrum of a signal. In the **Configure Spectral Measurements** dialog box that displays, select the settings shown in the following figure:



- (1) Click the **OK** button to close the dialog box.
- (2) Resize the Spectral Measurements Express VI to display the **error out** output.
- (3) Right-click the **Power Spectrum** output and select **Create»Graph Indicator** from the shortcut menu.



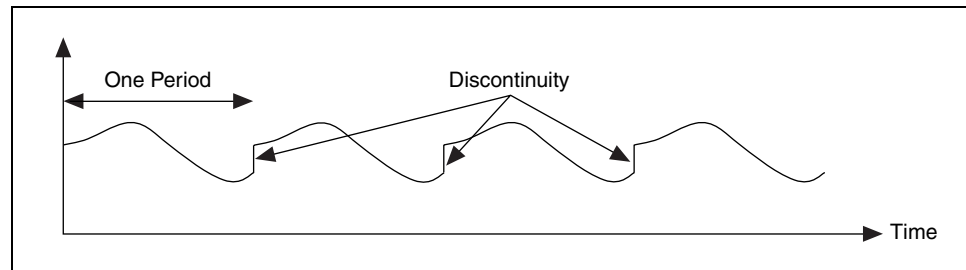
- c. Place the Or function, located on the **Functions»Arithmetic & Comparison»Express Boolean** palette, on the block diagram.
- d. Place the Unbundle by Name function, located on the **Functions»All Functions»Cluster** palette, on the block diagram. This function returns the error **status** Boolean value. If an error occurs, the While Loop stops.

4. Save the VI as `Power Spectrum Express.vi` in the `C:\Exercises\LabVIEW DAQ` directory.
5. Run the VI.
6. Change the frequency that the Simulate Signal Express VI generates. Notice how the peak moves on the spectrum plot.
7. Save and close the VI when you finish.

End of Exercise 6-1

D. About Spectral Leakage and Smoothing Windows

In practical applications, you can obtain only a finite number of samples of the signal. When you use the DFT/FFT to find the frequency content of a signal, the data is assumed to be a single period of a periodically repeating waveform, as shown in the following illustration. The first period shown is the one sampled. The waveform that corresponds to this period is then repeated in time to produce the periodic waveform.



Because of the assumption of periodicity of the waveform, discontinuities between successive periods occur. This happens when you sample a noninteger number of cycles. These artificial discontinuities appear as very high frequencies in the spectrum of the signal, frequencies not present in the original signal. These frequencies can be higher than the Nyquist frequency, and as you have seen before, will be aliased somewhere between 0 and $f_s/2$. The spectrum you get by using the DFT/FFT therefore will not be the actual spectrum of the original signal but will be a “smeared” version. It appears as if the energy at one frequency has “leaked out” into all the other frequencies. This phenomenon is known as spectral leakage.

Figure 6-1 shows a sine wave and its corresponding Fourier transform. The sampled time-domain waveform is shown in Graph 1. Because the Fourier transform assumes periodicity, you repeat this waveform in time. The periodic time waveform of the sine wave of Graph 1 is shown in Graph 2. The corresponding spectral representation is shown in Graph 3. Because the time record in Graph 2 is periodic, with no discontinuities, its spectrum is a single line that shows the frequency of the sine wave. The reason that the waveform in Graph 2 does not have any discontinuities is because you sampled an integer number of cycles—in this case, one cycle—of the time waveform.

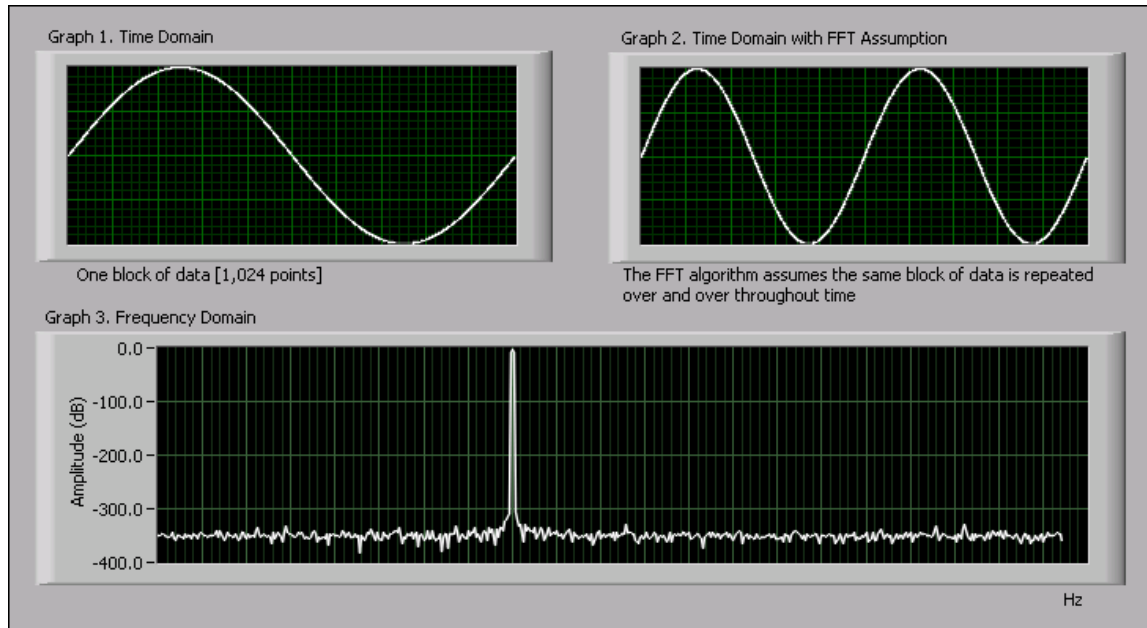


Figure 6-1. Sine Wave and Corresponding Fourier Transform

In Figure 6-2, you see the spectral representation when you sample a noninteger number of cycles of the time waveform, namely 1.25. Graph 1 now consists of 1.25 cycles of the sine wave. When you repeat this periodically, the resulting waveform as shown in Graph 2 consists of discontinuities. The corresponding spectrum is shown in Graph 3, in which the energy is spread over a wide range of frequencies. This smearing of the energy is spectral leakage. The energy leaked out of one of the FFT lines and smeared itself into all the other lines.

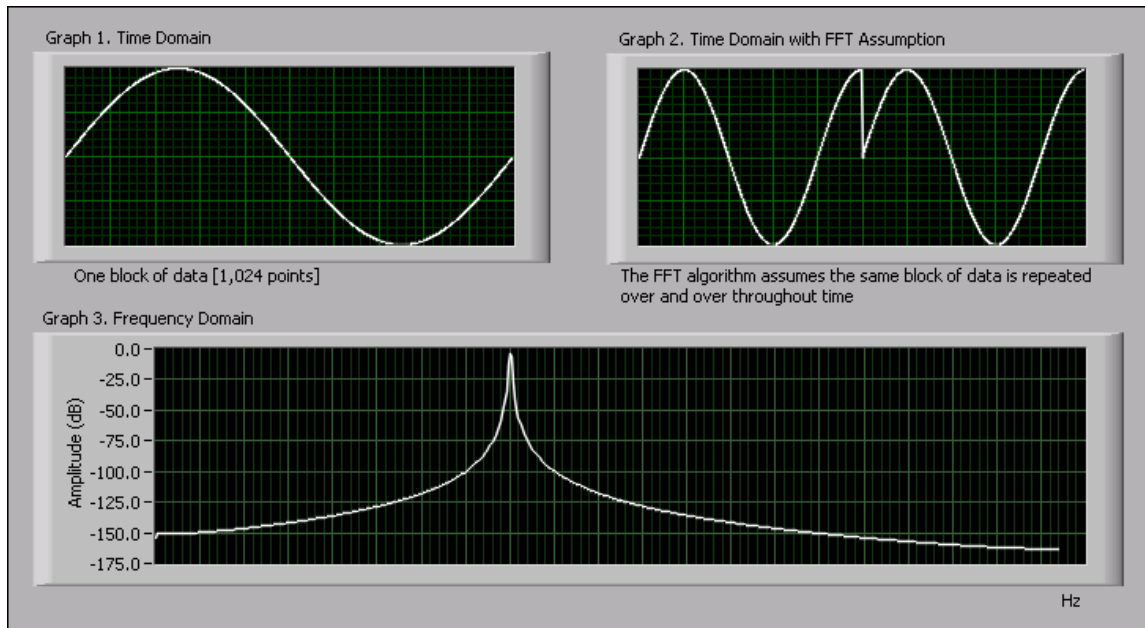


Figure 6-2. Spectral Representation When Sampling a Nonintegral Number of Samples

Leakage exists because of the finite time record of the input signal. One solution for overcoming leakage is to take an infinite time record, from $-\infty$ to $+\infty$, then have the FFT calculate one single line at the correct frequency. Because you are limited to having a finite time record, another technique known as windowing is used to reduce the spectral leakage.

The amount of spectral leakage depends on the amplitude of the discontinuity. The larger the discontinuity, the more leakage, and vice versa. You can use windowing to reduce the amplitude of the discontinuities at the boundaries of each period. Windowing consists of multiplying the time record by a finite-length window whose amplitude varies smoothly and gradually toward zero at the edges. This is shown in Figure 6-3, where the original time signal is windowed using a Hamming window. The time waveform of the windowed signal gradually tapers to zero at the ends. When performing Fourier or spectral analysis on finite-length data, you can use windows to minimize the transition edges of the sampled waveform. A smoothing window function applied to the data before it is transformed into the frequency domain minimizes spectral leakage.

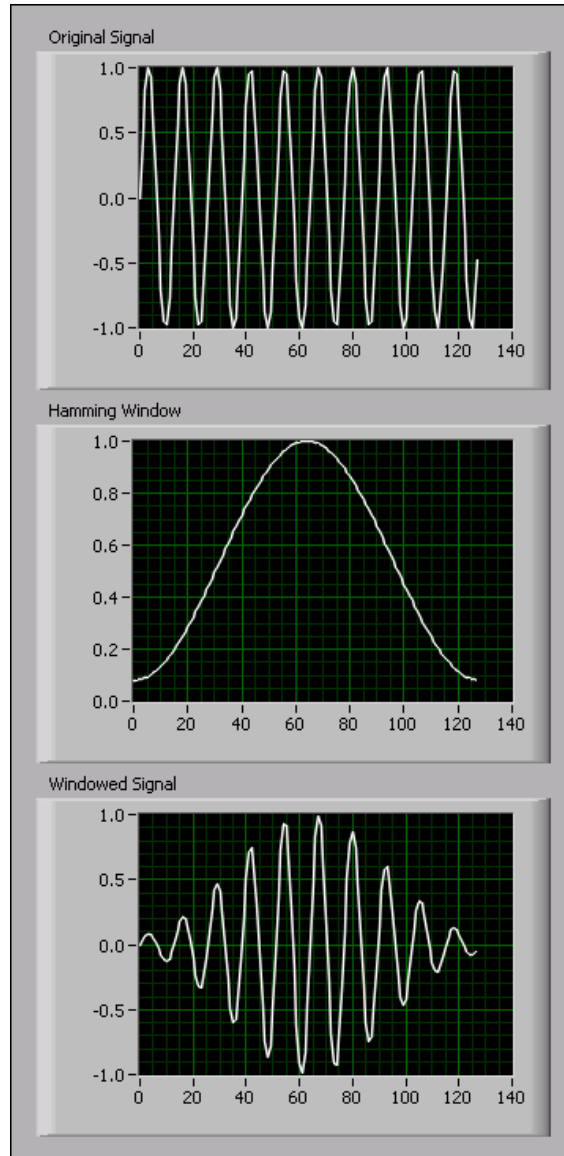


Figure 6-3. Time Signal Windowed Using a Hamming Window

If the time record contains an integral number of cycles, as shown in Figure 6-1, the assumption of periodicity does not result in any discontinuities, and therefore no spectral leakage occurs. The problem arises only when you have a nonintegral number of cycles.

There are several reasons to use windowing:

- Defining the duration of the observation
- Reducing spectral leakage
- Separating a small amplitude signal from a larger amplitude signal with frequencies very close to each other

E. Characteristics of Different Types of Window Functions

Applying a window to, or windowing, a signal in the time domain is equivalent to multiplying the signal by the window function. Because multiplication in the time domain is equivalent to convolution in the frequency domain, the spectrum of the windowed signal is a convolution of the spectrum of the original signal with the spectrum of the window. Windowing changes the shape of the signal in the time domain and affects the spectrum that you see.

Depending on the application, one window may be more useful than the others. Using the Spectral Measurements Express VI, you can choose from rectangular (none), Hanning, Hamming, Blackman-Harris, Exact Blackman, Blackman, Flat Top, 4 Term B-Harris, 7 Term B-Harris, and Low Sidelobe windows. The Exponential, General Cosine, Cosine Tapered, Force, Kaiser-Bessel and Triangle windowing functions are also available in the **Analyze»Signal Processing»Windows** palette.

Rectangular (None)

The rectangular window has a value of one over its time interval. Mathematically, it can be written as

$$w[n] = 1.0 \text{ for } n = 0, 1, 2, \dots, N - 1$$

where N is the length of the window. Applying a rectangular window is equivalent to not using any window because the rectangular function truncates the signal to within a finite time interval. The rectangular window has the highest amount of spectral leakage. Figure 6-4 shows the rectangular window for $N = 32$.

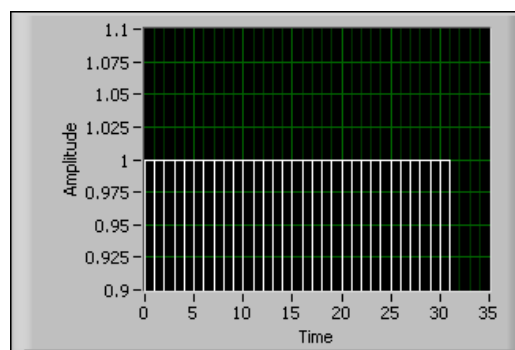


Figure 6-4. Rectangular Window

The rectangular window is useful for analyzing transients that have a duration shorter than that of the window. It also is used in order tracking, where the sampling frequency is adjusted depending on the speed of the

shaft of a machine. In this application, it detects the main mode of vibration of the machine and its harmonics.

Hanning

The Hanning window has a shape similar to that of a half-cycle of a cosine wave. Its defining equation is

$$w[n] = 0.5 - 0.5 \cos(2\pi n/N) \text{ for } n = 0, 1, 2, \dots, N - 1$$

Figure 6-5 shows a Hanning window with $N = 32$.

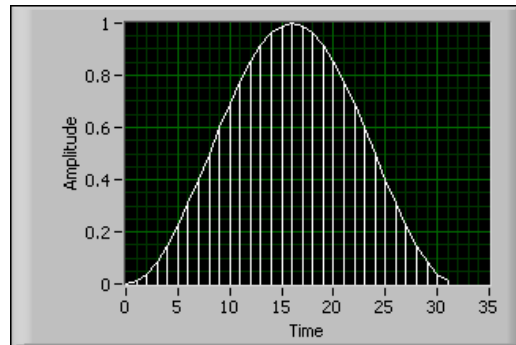


Figure 6-5. Hanning Window

The Hanning window is useful for analyzing transients longer than the time duration of the window and for general purpose applications.

Hamming

The Hamming window is a modified version of the Hanning window. Its shape also is similar to that of a cosine wave. It can be defined as:

$$w[n] = 0.54 - 0.46 \cos(2\pi n/N) \text{ for } n = 0, 1, 2, \dots, N - 1$$

Figure 6-6 shows a Hamming window with $N = 32$.

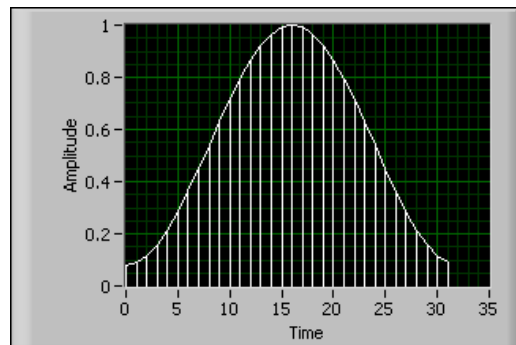


Figure 6-6. Hamming Window

The Hanning and Hamming windows are somewhat similar. In the time domain, the Hamming window does not get as close to 0 near the edges as the Hanning window.

Blackman–Harris

The Blackman-Harris window, as well as the Hanning window, is useful for measuring very low-level components in the presence of a large input signal, such as a distortion measurement.

The Blackman-Harris window applies a three-term window to the input signal. It can be defined as:

$$w[n] = 0.422323 - 0.49755 \cos(2\pi n/N) + 0.07922 \cos(4\pi n/N)$$

for $n = 0, 1, 2, \dots, N - 1$

Figure 6-7 shows a Blackman-Harris window with $N = 32$.

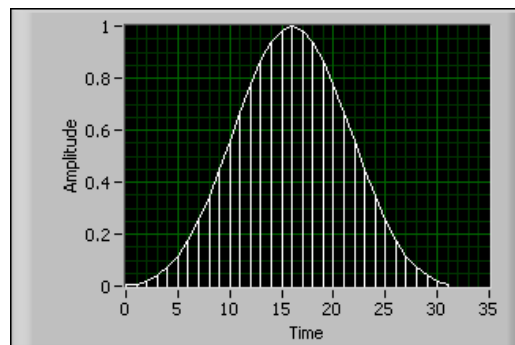


Figure 6-7. Blackman-Harris Window

Exact Blackman

The Exact Blackman window is similar to the Blackman-Harris window, but with less taper at the end.

It can be defined as:

$$w[n] = [a_0 - a_1 \cos(2\pi n/N) + a_2 \cos(4\pi n/N)]$$

for $n = 0, 1, 2, \dots, N - 1$

$$a_0 = 7938/18608$$

$$a_1 = 9240/18608$$

$$a_2 = 1430/18608$$

Figure 6-8 shows an Exact Blackman window with $N = 32$.

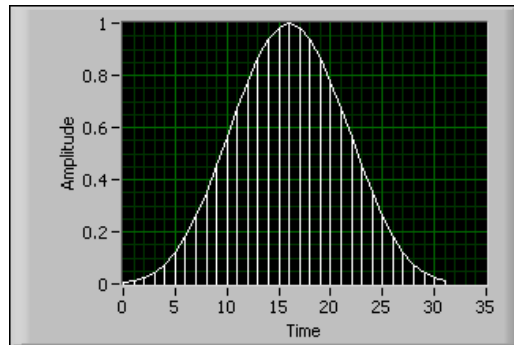


Figure 6-8. Exact Blackman Window

Blackman

The Blackman window is similar to the Hanning and Hamming window, but has an additional cosine term that further reduces the ripple effect.

It can be defined as:

$$w[n] = 0.42 - 0.5 \cos(2\pi n/N) + 0.08 \cos(4\pi n/N)$$

for $n = 0, 1, 2, \dots, N - 1$

Figure 6-9 shows a Blackman window with $N = 32$.

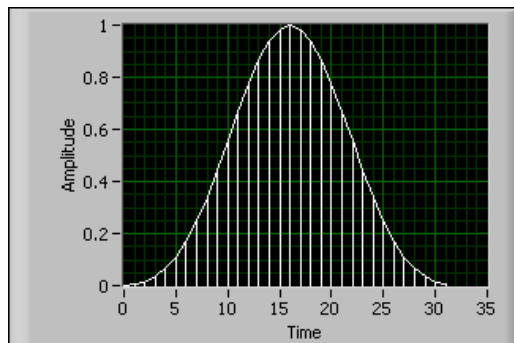


Figure 6-9. Blackman Window

Flat Top

The Flat Top window consists of more cosine terms than any other window we have studied thus far. The second harmonic term causes the window to drop below zero.

It can be defined as:

$$w[n] = 0.21557895 - 0.41663158 \cos(2\pi n/N) + 0.277263158 \cos(4\pi n/N) - 0.083578947 \cos(6\pi n/N) + 0.006947368 \cos(8\pi n/N)$$

for $n = 0, 1, 2, \dots, N - 1$

Figure 6-10 shows a Flat Top window with $N = 32$.

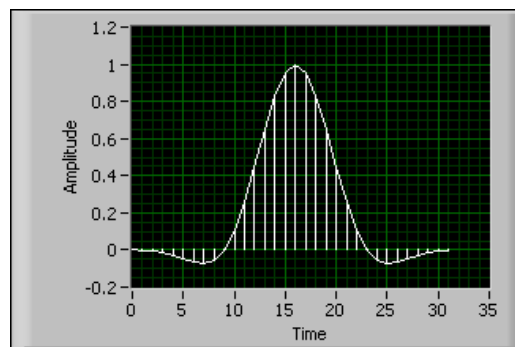


Figure 6-10. Flat Top Window

4 Term B-Harris

The 4 Term B-Harris window is an extension of the Blackman-Harris window, adding an additional cosine term.

It can be defined as:

$$w[n] = 0.35875 - 0.48829 \cos(2\pi n/N) + 0.14128 \cos(4\pi n/N) - 0.01168 \cos(6\pi n/N)$$

for $n = 0, 1, 2, \dots, N - 1$

Figure 6-11 shows a 4 Term B-Harris window with $N = 32$.

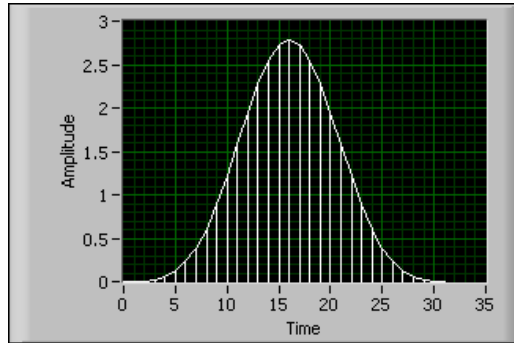


Figure 6-11. 4 Term B-Harris Window

7 Term B-Harris

The 7 Term B-Harris window is an extension of the Blackman-Harris window, adding an additional four cosine terms.

It can be defined as:

$$w[n] = 0.27105 - 0.43329 \cos(2\pi n/N) + 0.21812 \cos(4\pi n/N) \\ - 0.06593 \cos(6\pi n/N) + 0.01081 \cos(8\pi n/N) \\ - 7.7658E-4 \cos(10\pi n/N) + 1.3887E-5 \cos(12\pi n/N)$$

for $n = 0, 1, 2, \dots, N - 1$

Figure 6-12 shows a 7 Term B-Harris window with $N = 32$.

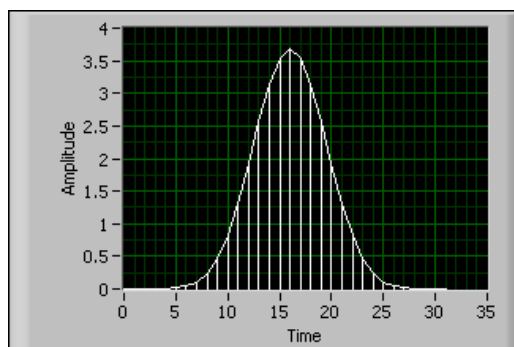


Figure 6-12. 7 Term B-Harris Window

Low Sidelobe

The Low Sidelobe window further reduces the size of the mainlobe.

It can be defined as:

$$w[n] = 0.323215218 - 0.471492057 \cos(2\pi n/N) + 0.17553428 \cos(4\pi n/N) \\ - 0.028497078 \cos(6\pi n/N) + 0.001261367 \cos(8\pi n/N)$$

for $n = 0, 1, 2, \dots, N - 1$

Figure 6-13 shows a Low Sidelobe window with $N = 32$.

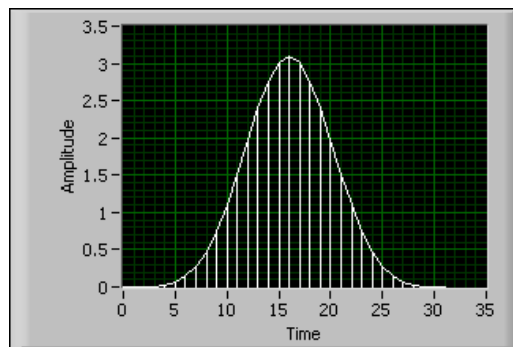


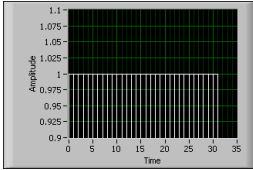
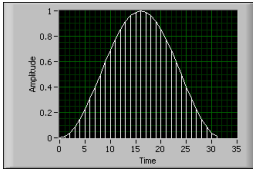
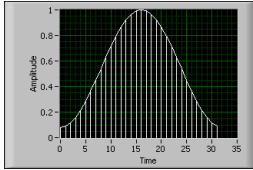
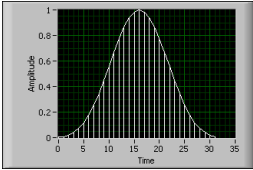
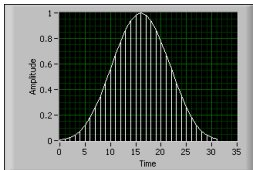
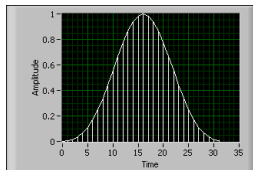
Figure 6-13. Low Sidelobe Window

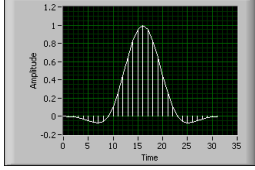
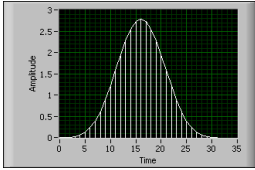
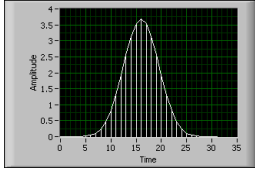
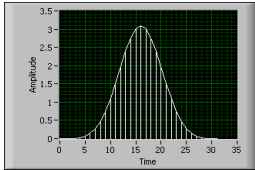
F. Determining Which Type of Window To Use

The type of window you choose depends on the type of signal you have and what you are looking for. Selecting the correct window requires knowledge of the signal that you are analyzing. The following table shows the different types of signals and the windows you can use with them.

Type of signal	Window
Transients whose duration is shorter than the length of the window	Rectangular
Transients whose duration is longer than the length of the window	Hanning
General purpose applications	Hanning
Order tracking	Rectangular
System analysis (frequency response measurements)	Hanning (for random excitation), rectangular (for pseudorandom excitation)
Separation of two tones with frequencies very close to each other but with widely differing amplitudes	Kaiser-Bessel
Separation of two tones with frequencies very close to each other but with almost equal amplitudes	Rectangular

If you do not have sufficient prior knowledge of the signal, experiment with different windows to find the best one.

Window	Equation	Shape	Applications
Rectangular (None)	$w[n] = 1.0$		Detecting transients whose duration is shorter than the length of the window; order tracking; separating two tones with frequencies and amplitudes very close to each other; system response
Hanning	$w[n] = 0.5 - 0.5 \cos\left(\frac{2\pi n}{N}\right)$		General purpose applications; system analysis; transients whose duration is longer than the length of the window
Hamming	$w[n] = 0.54 - 0.46 \cos\left(\frac{2\pi n}{N}\right)$		—
Blackman-Harris	$w[n] = 0.422322 -$ $0.49755 \cos\left(\frac{2\pi n}{N}\right) +$ $0.07922 \cos\left(\frac{4\pi n}{N}\right)$		Similar to Blackman
Exact Blackman	$w[n] = \frac{7938}{18608} -$ $\frac{9240}{18608} \cos\left(\frac{2\pi n}{N}\right) +$ $\frac{1430}{18608} \cos\left(\frac{4\pi n}{N}\right)$		Similar to Blackman
Blackman	$w[n] = 0.42 - 0.5 \cos\left(\frac{2\pi n}{N}\right) +$ $0.8 \cos\left(\frac{4\pi n}{N}\right)$		Transient signals; similar to Hanning and Hamming windows but adds one additional cosine term to reduce ripple

Window	Equation	Shape	Applications
Flat Top	$w[n] = 0.21557895 - 0.41663158 \cos\left(\frac{2\pi n}{N}\right) + 0.277263158 \cos\left(\frac{4\pi n}{N}\right) - 0.083578947 \cos\left(\frac{6\pi n}{N}\right) + 0.006947368 \cos\left(\frac{8\pi n}{N}\right)$		Accurate single tone amplitude measurements when there are no nearby frequency components
4 Term B-Harris	$w[n] = 0.35875 - 0.48829 \cos\left(\frac{2\pi n}{N}\right) + 0.14128 \cos\left(\frac{4\pi n}{N}\right) - 0.01168 \cos\left(\frac{6\pi n}{N}\right)$		Similar to Blackman
7 Term B-Harris	$w[n] = 0.27105 - 0.43329 \cos\left(\frac{2\pi n}{N}\right) + 0.21812 \cos\left(\frac{4\pi n}{N}\right) - 0.06593 \cos\left(\frac{6\pi n}{N}\right) + 0.01081 \cos\left(\frac{8\pi n}{N}\right) - 7.7658E-4 \cos\left(\frac{10\pi n}{N}\right) + 1.3887E-5 \cos\left(\frac{12\pi n}{N}\right)$		Similar to Blackman
Low Sidelobe	$w[n] = 0.323215218 - 0.471492057 \cos\left(\frac{2\pi n}{N}\right) + 0.17553428 \cos\left(\frac{4\pi n}{N}\right) - 0.028497078 \cos\left(\frac{6\pi n}{N}\right) + 0.001261367 \cos\left(\frac{8\pi n}{N}\right)$		—

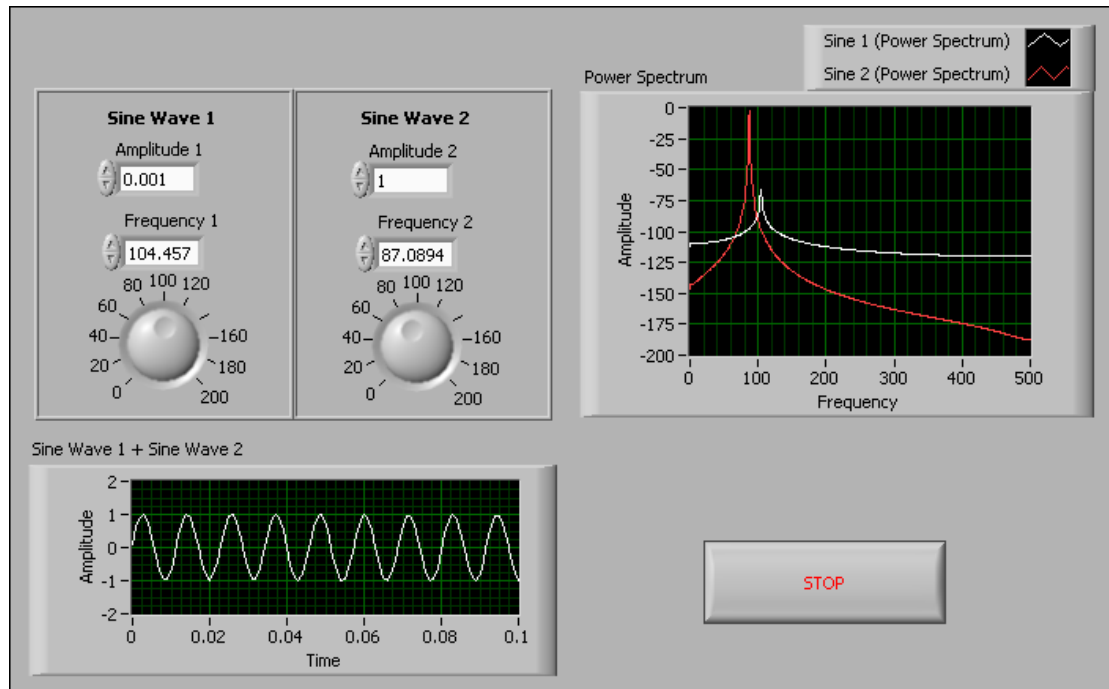
Exercise 6-2 Window Comparison VI

Objective: To build a VI that separates two sine waves of almost the same frequency, but widely differing amplitudes.

In this exercise, two sine waves of different amplitudes are added together and transformed into the frequency domain. Sine wave 1 has a much smaller amplitude than sine wave 2. Without windowing, it is not possible to distinguish between the two sine waves in the frequency domain. With an appropriate window, you can clearly separate the peaks in the frequency domain that corresponds to the two sine waves. The frequency-domain plot shows the results so you can compare the effect of different window functions.

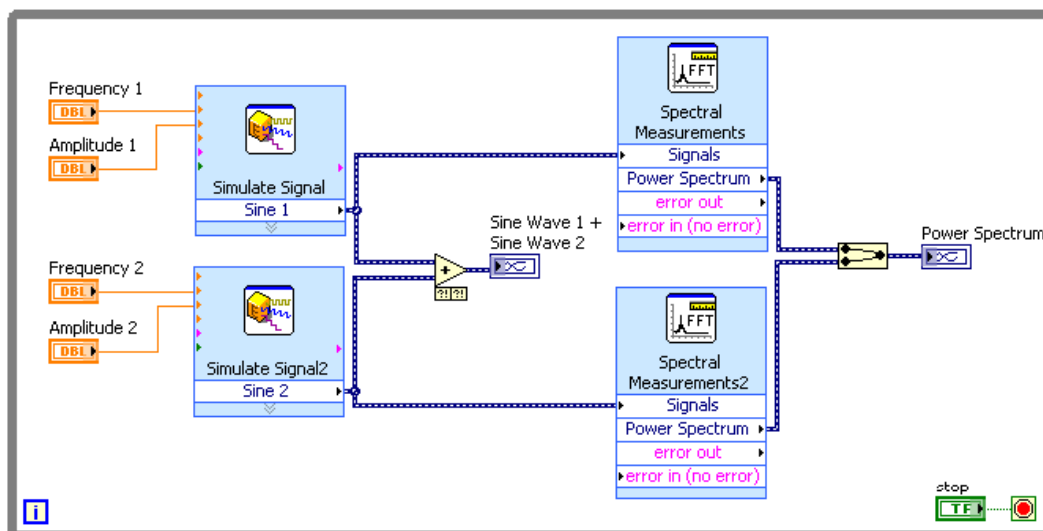
Front Panel

1. Open a blank VI and build the following front panel.



Block Diagram

2. Build the following block diagram.



- a. Place the Simulate Signal Express VI, located on the **Functions»Signal Analysis** palette, on the block diagram. This Express VI to generate a signal of the specified type, in this case, a sine wave. In the **Configure Simulate Signals** dialog box that displays, set the **Samples per Second (Hz)** to 1000, remove the checkmark from the **Automatic** checkbox, and enter 1000 for the **Number of Samples**. Leave all other settings at their default values. Click the **OK** button to close the dialog box.



- b. Place the Spectral Measurements Express VI, located on the **Functions»Signal Analysis** palette, on the block diagram. This Express VI performs an FFT on an input signal. In the **Configure Spectral Measurements** dialog box that appears, select **Power Spectrum** set **Result** to **dB**, select **None** from the **Window** pull-down menu, and remove the checkmark from the **Averaging** checkbox. Click the **OK** button to close the dialog box.



- c. Place the Merge Signals function, located on the **Functions»Signal Manipulation** palette, on the block diagram. This function merges two or more signals into a single output. Wire the output of the Merge Signals function to the Power Spectrum graph.
3. Save the VI as `Window Comparison.vi` in the `C:\Exercises\LabVIEW DAQ` directory.
 4. Return to the front panel. Using the numeric controls, set the amplitude of Sine Wave 1 to 0.001, and the amplitude of Sine Wave 2 to 1.000. With the knob controls, set the frequency of Sine Wave 1 near 70 and the frequency of Sine Wave 2 near 60. Adjusting the frequency of Sine Wave 2 using the knob control causes the smaller amplitude to be closer to the larger amplitude in the frequency-domain plot.

5. Notice in the graph that when the frequency of the smaller amplitude signal (Sine Wave 1) is closer to that of the larger amplitude signal (Sine Wave 2), the peak that corresponds to the smaller signal is not detected. Applying a window function is the only way to detect the smaller signal. The discontinuity causes the spectrum to spread out. Signals at smaller amplitudes are lost in the sidelobes of the larger amplitude signal.
6. Stop the VI and return to the block diagram. Double-click each of the Spectral Measurements Express VIs and select different window functions.
7. Return to the front panel and run the VI.
8. Compare different window functions by choosing another window for the Spectral Measurements Express VIs. Which one(s) can distinguish between the two frequency components?
9. Stop and close the VI when you finish.

End of Exercise 6-2

G. Filtering

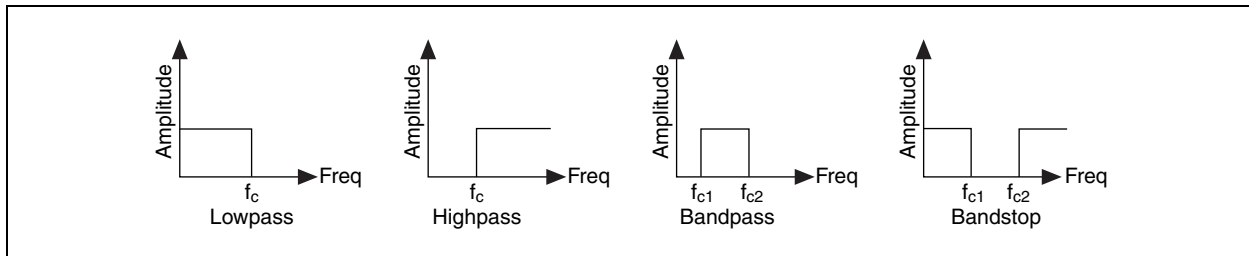
Filtering is the process by which the frequency content of a signal is altered. It is one of the most commonly used signal processing techniques. Common everyday examples of filtering are the bass and treble controls on a stereo system. The bass control alters the low-frequency content of a signal, and the treble control alters the high-frequency content. By varying these controls, you are actually filtering the audio signal. Some other applications where filtering is useful are removing noise and performing decimation (lowpass filtering the signal and reducing the sample rate).

H. Ideal Filters

Filters alter or remove unwanted frequencies. Depending on the frequency range that they either pass or attenuate, they can be classified into the following types:

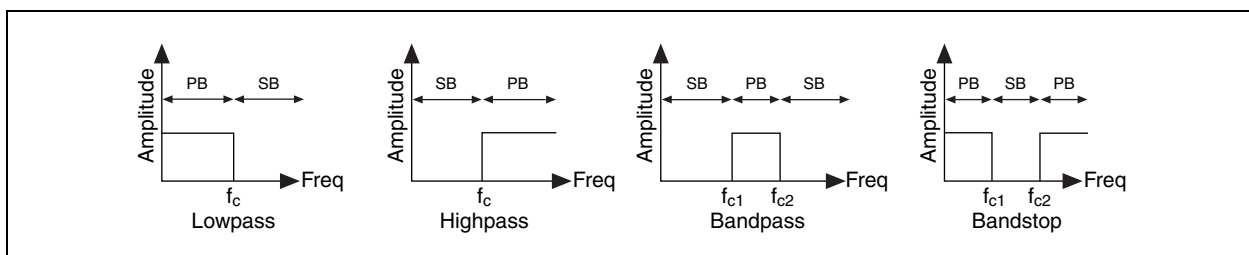
- A lowpass filter passes low frequencies but attenuates high frequencies.
- A highpass filter passes high frequencies but attenuates low frequencies.
- A bandpass filter passes a certain band of frequencies.
- A bandstop filter attenuates a certain band of frequencies.

The following illustration shows the ideal frequency response of these filters.



The lowpass filter passes all frequencies below f_c , and the highpass filter passes all frequencies above f_c . The bandpass filter passes all frequencies between f_{c1} and f_{c2} , and the bandstop filter attenuates all frequencies between f_{c1} and f_{c2} . The frequency points f_c , f_{c1} , and f_{c2} are known as the cutoff frequencies of the filter. When designing filters, you need to specify these cutoff frequencies.

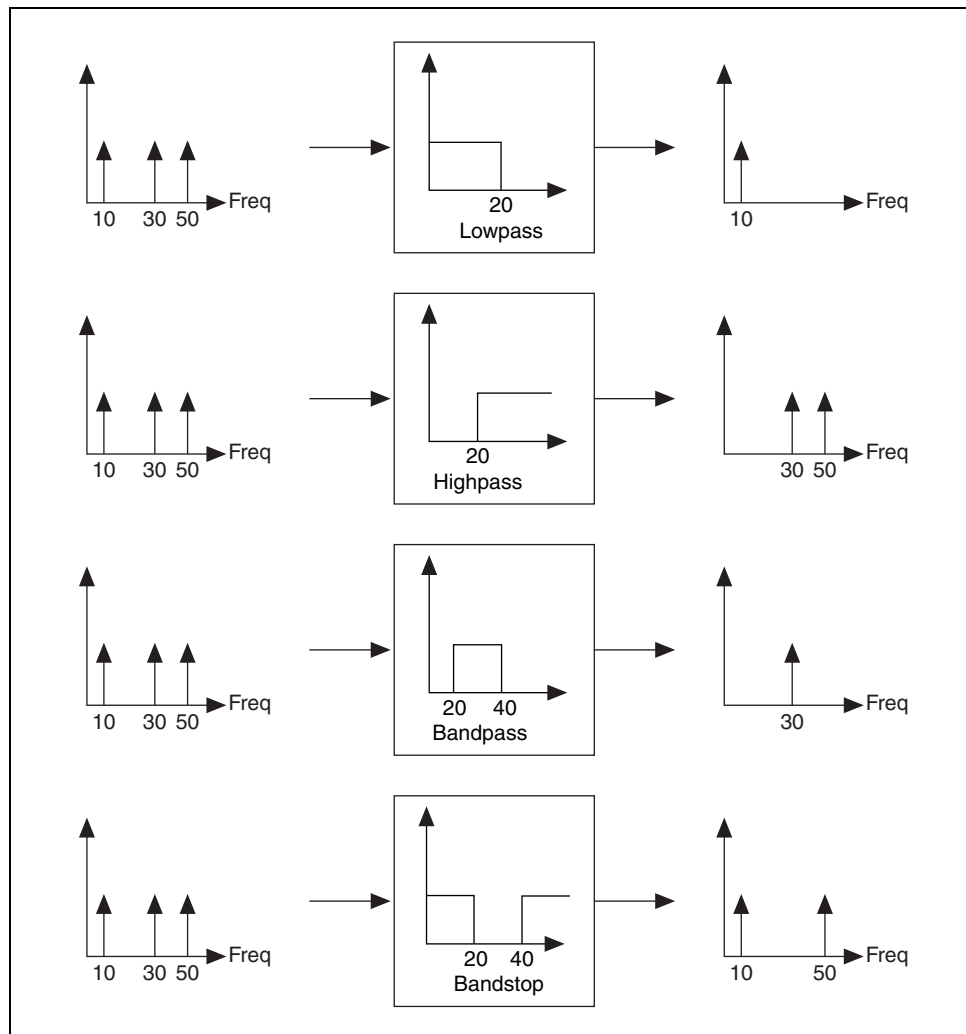
The frequency range that is passed through the filter is known as the passband (PB) of the filter. An ideal filter has a gain of one (0 dB) in the passband so that the amplitude of the signal neither increases nor decreases. The stopband (SB) corresponds to that range of frequencies that do not pass through the filter at all and are rejected (attenuated). The following illustration shows the passband and the stopband for the different types of filters.



The lowpass and highpass filters have one passband and one stopband, the bandpass filter has one passband but two stopbands, and the bandstop filter has two passbands but one stopband.

How Filters Affect Signal Frequency Content

A signal that contains frequencies of 10 Hz, 30 Hz, and 50 Hz is passed through lowpass, highpass, bandpass, and bandstop filters. The lowpass and highpass filters have a cutoff frequency of 20 Hz, and the bandpass and bandstop filters have cutoff frequencies of 20 Hz and 40 Hz. The following illustration shows the output of the filter in each case.

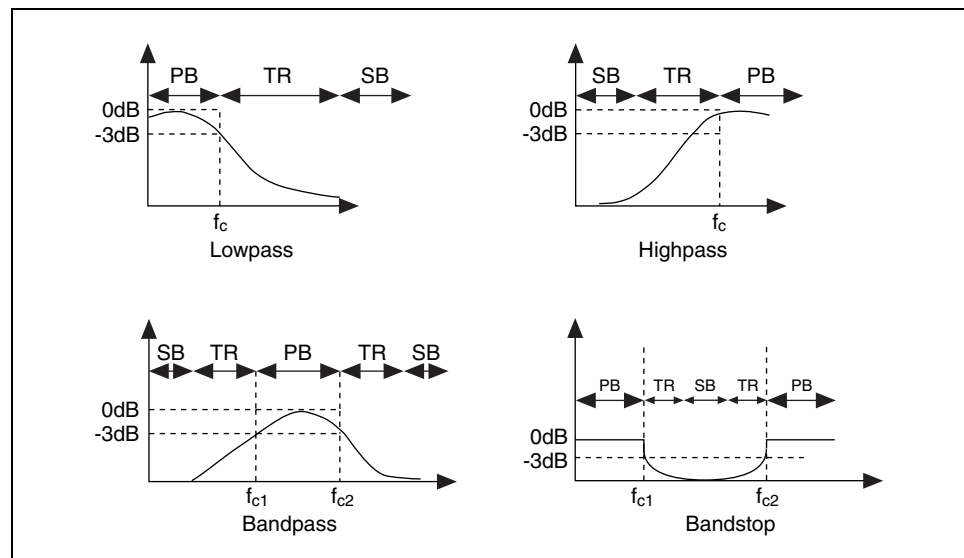


I. Practical (Nonideal) Filters

Ideally, a filter should have a unit gain (0 dB) in the passband, and a gain of zero ($-\infty$ dB) in the stopband. However, in a real implementation, not all of these criteria can be fulfilled. In practice, there is always a finite transition region between the passband and the stopband. In this region, the gain of the filter changes gradually from one (0 dB) in the passband to zero ($-\infty$ dB) in the stopband.

Transition Band

A filter should have a unit gain (0 dB) in the passband and a gain of zero ($-\infty$ dB) in the stopband. In a real implementation, not all of these criteria can be fulfilled. A finite transition region always occurs between the passband and the stopband. In this region, the gain of the filter changes gradually from 1 (0 dB) in the passband to 0 ($-\infty$) in the stopband. The following illustration shows the passband, the stopband, and the transition region (TR) for the different types of nonideal filters. The passband becomes the frequency range within which the gain of the filter varies from 0 dB to -3 dB. Although the -3 dB range is most common, depending on the application, other values (-0.5 dB, -1 dB, and so on) also might be used.



Passband Ripple and Stopband Attenuation

In many applications, it is acceptable to allow the gain in the passband to vary slightly from unity. The variation in the passband is called the passband ripple, which is the difference between the actual gain and the desired gain of unity. The stopband attenuation cannot be infinite, and you must specify the value you want. Both the passband ripple and the stopband attenuation are measured in decibels or dB, defined by:

$$\text{dB} = 20 \cdot \log_{10}(A_o(f)/A_i(f))$$

where \log_{10} denotes the logarithm to the base 10, and $A_i(f)$ and $A_o(f)$ are the amplitudes of a particular frequency f before and after the filtering, respectively.

For example, for -0.02 dB passband ripple, the formula gives

$$-0.02 = 20 \cdot \log_{10}(A_o(f)/A_i(f))$$

$$A_o(f)/A_i(f) = 10^{-0.001} = 0.9977$$

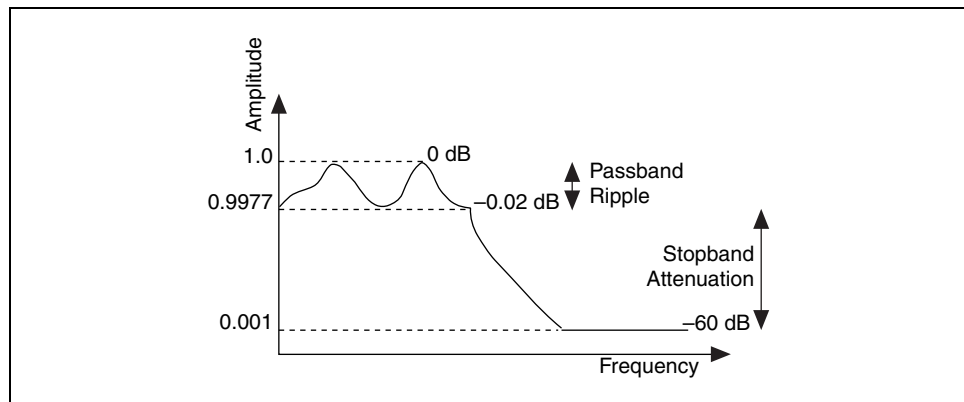
which shows that the ratio of input and output amplitudes is close to unity.

If you have -60 dB attenuation in the stopband, you have

$$-60 = 20 \cdot \log_{10}(A_o(f)/A_i(f))$$

$$A_o(f)/A_i(f) = 10^{-3} = 0.001$$

which means the output amplitude is 1/1000 of the input amplitude. The following illustration, though not drawn to scale, illustrates this concept.



Note Attenuation is usually expressed in decibels without the word minus, but a negative dB value is normally assumed.

J. Advantages of Digital Filters over Analog Filters

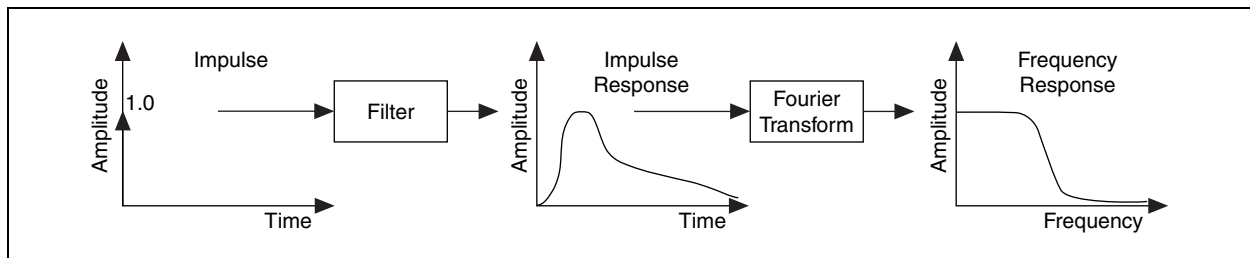
An analog filter has an analog signal at both its input and its output. Both the input, $x(t)$, and output, $y(t)$, are functions of a continuous variable t and can take on an infinite number of values. Analog filter design is about 50 years older than digital filter design. Many analog filter design books featuring simple, well-tested filter designs are available. However, this type of filter design is often reserved for specialists because it requires advanced mathematical knowledge and understanding of the processes involved in the system that affects the filter.

Modern sampling and digital signal processing tools make it possible to replace analog filters with digital filters in applications that require flexibility and programmability. These applications include audio, telecommunications, geophysics, and medical monitoring. The following list explains the advantages of digital filters:

- They are software programmable and therefore easy to build and test.
- They require only the arithmetic operations of multiplication and addition/subtraction, so they are easier to implement.
- They are stable (they do not change with time or temperature) and predictable.
- They do not drift with temperature or humidity or require precision components.
- They have a superior performance-to-cost ratio.
- They do not suffer from manufacturing variations or aging.

K. IIR and FIR Filters

Another method of classification of filters is based on their impulse response. An impulse response is the response of a filter to an input that is an impulse ($x[0] = 1$ and $x[i] = 0$ for all $i \neq 0$), as shown in the following illustration. The Fourier transform of the impulse response is known as the frequency response of the filter. The frequency response of a filter tells you what the output of the filter is going to be at different frequencies. The frequency response tells you the gain of the filter at different frequencies. For an ideal filter, the gain should be 1 in the passband and 0 in the stopband. All frequencies in the passband are passed without changes to the output, but there is no output for frequencies in the stopband.



If the impulse response of the filter falls to zero after a finite amount of time, it is known as a finite impulse response (FIR) filter. If the impulse response exists indefinitely, it is known as an infinite impulse response (IIR) filter. If the impulse response is FIR or IIR depends on how the output is calculated.

The basic difference between FIR and IIR filters is that for FIR filters, the output depends only on the current and past input values; for IIR filters, the output depends not only on the current and past input values, but also on the past output values.

As an example, consider a cash register at a supermarket. Let $x[k]$ be the cost of the k^{th} item that a customer buys, where $1 \leq k \leq N$, and N is the total number of items. The cash register adds the cost of each item to produce a running total. This running total $y[k]$, up to the k^{th} item, is given by

$$y[k] = x[k] + x[k-1] + x[k-2] + x[k-3] + \dots + x[1] \quad (6-2a)$$

Thus, the total for N items is $y[N]$. Because $y[k]$ is the total up to the k^{th} item, and $y[k-1]$ is the total up to the $(k-1)^{\text{st}}$ item, you can rewrite Equation 6-2a as

$$y[k] = y[k-1] + x[k] \quad (6-2b)$$

If you add a sales tax of 8%, Equations 6-2a and 6-2b yield

$$y[k] = 1.08x[k] + 1.08x[k - 1] + 1.08x[k - 2] + 1.08x[k - 3] + \dots + 1.08x[1] \quad (6-3a)$$

$$y[k] = y[k - 1] + 1.08x[k] \quad (6-3b)$$

Both Equations 6-3a and 6-3b are identical in describing the behavior of the cash register. The difference is that Equation 6-3a is implemented only in terms of the inputs, but Equation 6-3b is implemented in terms of both the input and the output. Equation 6-3a is known as the nonrecursive, or FIR, implementation. Equation 6-3b is known as the recursive, or IIR, implementation.

Filter Coefficients

In Equation 6-3a, the multiplying constant for each term is 1.08. In Equation 6-3b, the multiplying constants are 1 (for $y[k - 1]$) and 1.08 (for $x[k]$). These multiplying constants are known as the coefficients of the filter. For an IIR filter, the coefficients that multiply the inputs are forward coefficients, and those that multiply the outputs reverse coefficients.

Equations of the form 6-2a, 6-2b, 6-3a, or 6-3b that describe the operation of the filter are known as difference equations.

Advantages and Disadvantages of FIR and IIR Filters

The advantage of digital IIR filters over finite impulse response (FIR) filters is that IIR filters usually require fewer coefficients to perform similar filtering operations. Thus, IIR filters execute much faster and do not require extra memory because they execute in place.

The disadvantage of IIR filters is that the phase response is nonlinear. If the application does not require phase information, such as simple signal monitoring, IIR filters may be appropriate. Use FIR filters for applications that require linear phase responses. The recursive nature of IIR filters makes them difficult to design and implement.

L. Infinite Impulse Response Filters

IIR filters are digital filters whose output is calculated by adding a weighted sum of past output values with a weighted sum of past and current input values. Denoting the input values by $x[.]$ and the output values by $y[.]$, the general difference equation characterizing IIR filters is

$$a_0y[i] + a_1y[i-1] + a_2y[i-2] + \dots + a_{N_y-1}y[i-(N_y-1)] = b_0x[i] + b_1x[i-1] + b_2x[i-2] + \dots + b_{N_x-1}x[i-(N_x-1)]$$

$$a_0y[i] = -a_1y[i-1] - a_2y[i-2] - \dots - a_{N_y-1}y[i-(N_y-1)] + b_0x[i] + b_1x[i-1] + b_2x[i-2] + \dots + b_{N_x-1}x[i-(N_x-1)]$$

$$y[i] = \frac{1}{a_0} \left(- \sum_{j=1}^{N_y-1} a[j]y[i-j] + \sum_{k=0}^{N_x-1} b[k]x[i-k] \right) \quad (6-4)$$

where N_x is the number of forward coefficients ($b[k]$), and N_y is the number of reverse coefficients ($a[j]$). The output sample at the present sample index i is the sum of scaled present and past inputs ($x[i]$ and $x[i-k]$ when $j \neq 0$) and scaled past outputs ($y[i-j]$). Usually, N_x is equal to N_y , and this value is known as the order of the filter.



Note In all of the IIR filters implemented in LabVIEW, the coefficient a_0 is 1.

Practical IIR Filters

A lower order reduces arithmetic operations and therefore reduces computation error. A problem with higher order filtering is that you quickly run into precision errors with orders much greater than 20–30. This is the main reason for the cascade implementations over the direct form. Refer to the *LabVIEW Help* for more information about cascade form implementations. The orders of 1–20 are reasonable, with 30 being an upper limit. A higher order also means more filter coefficients and longer processing time.

The impulse response of the filter described by Equation 6-4 is of infinite length for nonzero coefficients. In practical filter applications, however, the impulse response of stable IIR filters decays to near zero in a finite number of samples.

In practice, the frequency response of filters differs from that of ideal filters. Depending on the shape of the frequency response, the IIR filters can be further classified into the following categories:

- Butterworth filters
- Chebyshev filters
- Chebyshev II or inverse Chebyshev filters
- Elliptic filters
- Bessel filters

In the following sections about each of these filters, the input signal is an impulse response. The signal is then filtered using the Filter Express VI and then the frequency response taken by using the Frequency Response Function (Mag-Phase) VI with no windowing applied.

Butterworth Filters

A Butterworth filter has no ripples in the passband or the stopband. Due to the lack of ripples, it is also known as the maximally flat filter. Its frequency response is characterized by a smooth response at all frequencies. Figure 6-14 shows the response of a lowpass Butterworth filter of different orders.

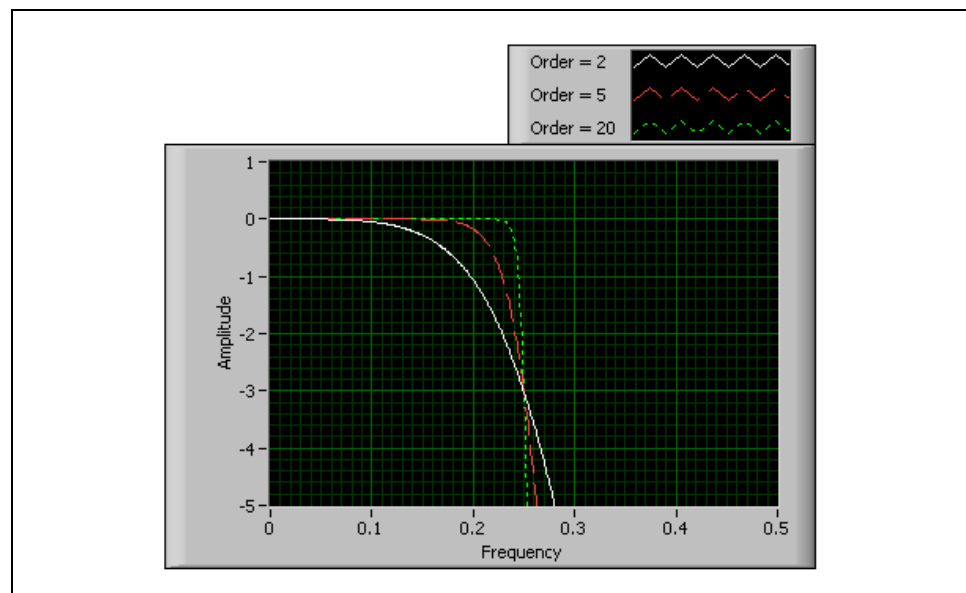


Figure 6-14. Butterworth Response

The region where the output of the filter is equal to 0 or very close to 0 is the passband of the filter. The region where the output approaches the negative amplitudes is the stopband. The region in between the passband and the stopband where the output gradually changes from 0 to negative amplitudes is the transition region.

The advantage of Butterworth filters is a smooth, monotonically decreasing frequency response in the transition region. As seen in Figure 6-14, the higher the filter order, the steeper the transition region.

Chebyshev Filters

The frequency response of Butterworth filters is not always an accurate approximation of the ideal filter response because of the slow rolloff between the passband, also known as the portion of interest in the spectrum, and the stopband, also known as the unwanted portion of the spectrum. Chebyshev filters have a smaller transition region than a Butterworth filter of the same order. However, this is achieved at the expense of ripples in the passband. You can use LabVIEW to specify the maximum amount of ripple (in dB) in the passband for a Chebyshev filter. The frequency response characteristics of Chebyshev filters have an equiripple (ripples all have the same magnitude) magnitude response in the passband, monotonically decreasing magnitude response in the stopband, and a sharper rolloff in the transition region as compared to Butterworth filters of the same order.

Figure 6-15 shows the response of a lowpass Chebyshev filter of different orders. In this graph, the y-axis scaling is in decibels. The steepness of the transition region increases with increasing order. The number of ripples in the passband also increases with increasing order.

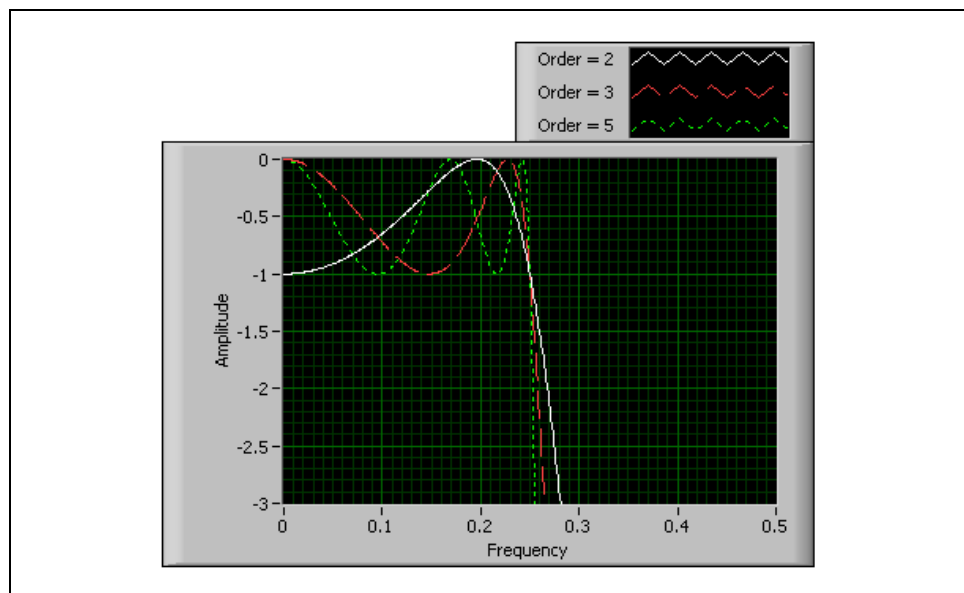


Figure 6-15. Chebyshev Response

The advantage of Chebyshev filters over Butterworth filters is the sharper transition between the passband and the stopband with a lower order filter. This produces smaller absolute errors and higher execution speeds.

Chebyshev II or Inverse Chebyshev Filters

Chebyshev II, also known as inverse Chebyshev or Type II Chebyshev filters, are similar to Chebyshev filters except that Chebyshev II filters have ripples in the stopband as opposed to the passband and are maximally flat in the passband as opposed to the stopband. For Chebyshev II filters, you can specify the amount of attenuation (in dB) in the stopband. The frequency response characteristics of Chebyshev II filters are equiripple magnitude response in the stopband, monotonically decreasing magnitude response in the passband, and a rolloff sharper than Butterworth filters of the same order. Figure 6-16 plots the response of a lowpass Chebyshev II filter of different orders.

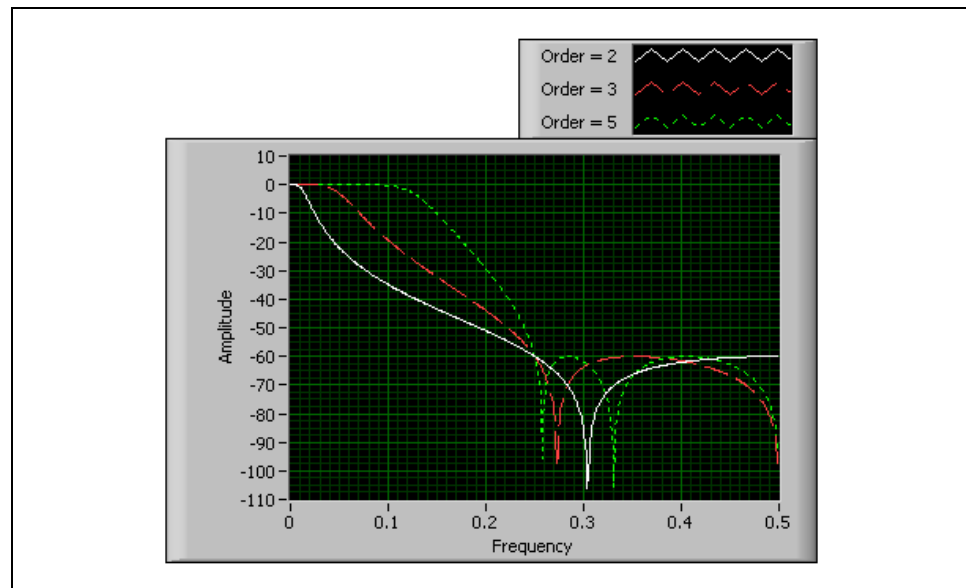


Figure 6-16. Chebyshev II Resonse

The advantage of Chebyshev II filters over Butterworth filters is that Chebyshev II filters give a sharper transition between the passband and the stopband with a lower order filter. This difference corresponds to a smaller absolute error and higher execution speed. One advantage of Chebyshev II filters over regular Chebyshev filters is that Chebyshev II filters have the ripples in the stopband instead of the passband.

Elliptic Filters

Chebyshev type I and II filters have a sharper transition region than a Butterworth filter of the same order. This is because they allowed ripples in the passband (type I) or the stopband (type II). Elliptic filters distribute the ripples over the passband and the stopband. Equiripples in the passband and the stopband characterize the magnitude response of elliptic filters. Compared with the same order Butterworth or Chebyshev filters, the elliptic design provides the sharpest transition between the passband and the stopband. For this reason, elliptic filters are quite popular in applications in which short transition bands are required and in which ripples can be tolerated. Figure 6-17 plots the response of a lowpass elliptic filter of different orders.

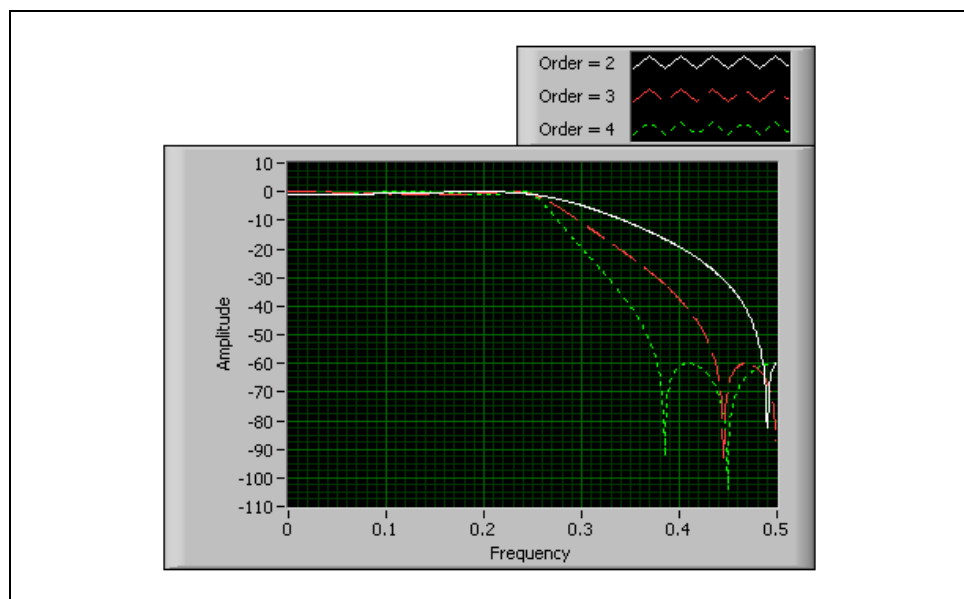


Figure 6-17. Elliptic Response

Notice the sharp transition edge for even low-order elliptic filters. For elliptic filters, you can specify the amount of ripple (in dB) in the passband and the attenuation (in dB) in the stopband.

Bessel Filters

The Bessel filter was designed with a square wave in mind and is thus ideal for digital filtering. Much like the Butterworth filter, the Bessel filter has a smooth passband and stopband response. Using the same filter order, the stopband attenuation of the Bessel filter is much lower than that of the Butterworth filter. Of all the filter types, the Bessel filter has the widest transition region for a fixed order. The main advantage of the Bessel filter is that the phase response is nearly linear throughout the passband.

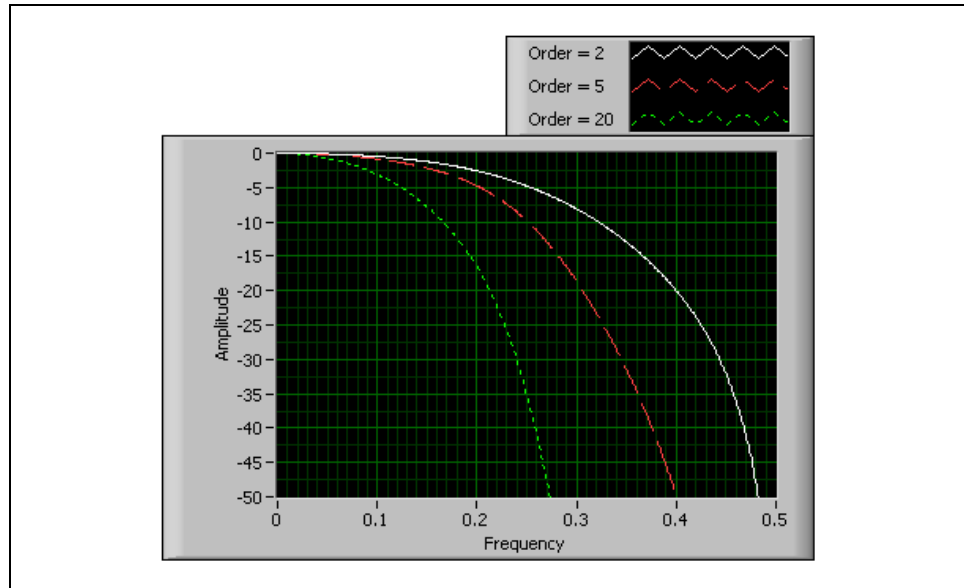
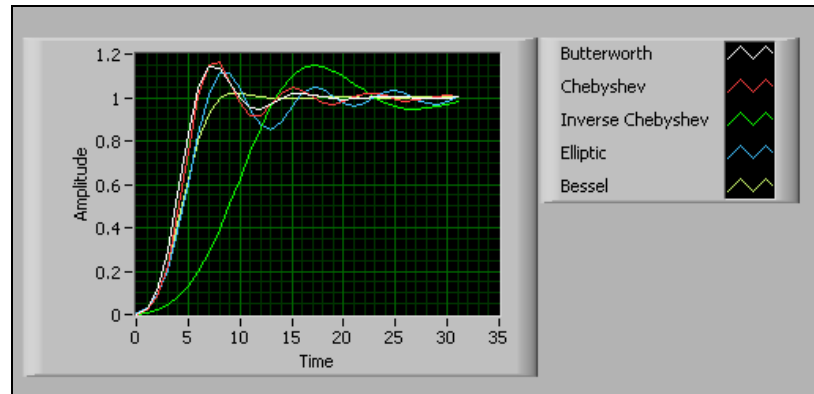


Figure 6-18. Bessel Response

M. IIR Filter Comparison

The following figure shows a comparison of the lowpass frequency responses for the five different IIR filter designs, all having the same order (five). The elliptic filter has the narrowest transition region, and the Bessel filter has the widest.



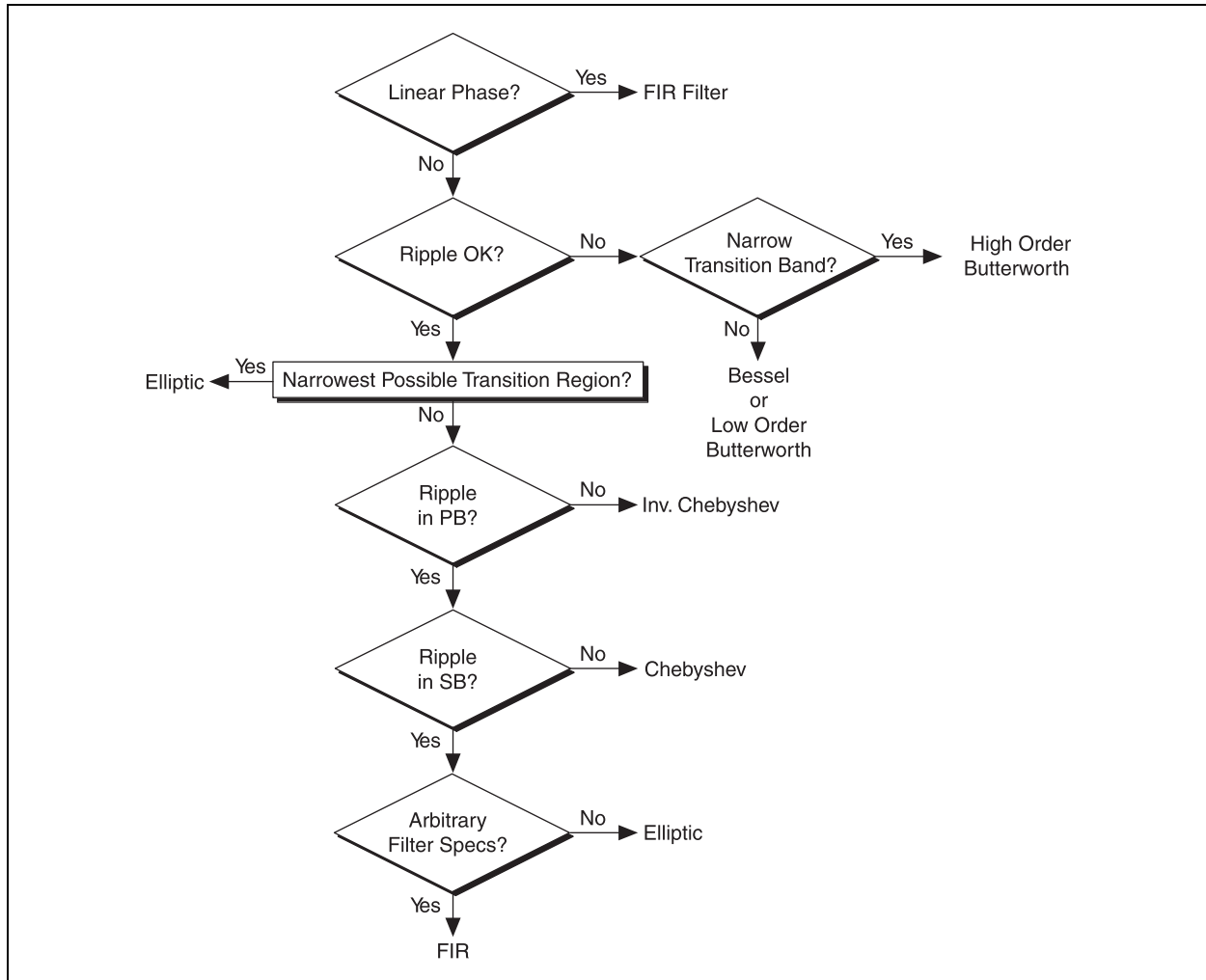
The following table compares the filter types.

IIR Filter Design	Response Characteristics	Width of Transition Region for a Fixed Order	Order Required for Given Filter Specifications
Butterworth	No ripples	—	—
Chebyshev	Ripple in PB	—	—
Inverse Chebyshev	Ripple in SB	—	—
Elliptic	Ripples in PB and SB	Narrowest	Lowest
Bessel	No ripples	Widest	Highest

The LabVIEW digital filter VIs handle all the design issues, computations, memory management, and actual data filtering internally and are transparent to the user. You do not need to be an expert in digital filters or digital filter theory to process the data. You need only to specify the control parameters such as the filter order, cutoff frequencies, amount of ripple, and stopband attenuation.

Deciding Which Filter to Use

Now that you have seen the different types of filters and their characteristics, the question arises as to which filter design is best suited for the application. Some of the factors affecting the choice of a suitable filter are if you require linear phase, if you can tolerate ripples, and if a narrow transition band is required. The following flowchart provides a guideline for selecting the correct filter. You might need to experiment with several different options before finding the best one.



N. Transient Response of IIR Filters

The output of a general IIR filter is given by

$$y[i] = -a_1y[i-1] - a_2y[i-2] - \dots - a_{N_y-1}y[i - (N_y - 1)] + b_0x[i] + b_1x[i-1] + b_2x[i-2] + \dots + b_{N_x-1}x[i - (N_x - 1)] \quad (6-5)$$

where N_x is the number of forward coefficients, N_y is the number of reverse coefficients, and a_0 is assumed to be equal to 1. With a second-order filter where $N_x = N_y = 2$, the corresponding difference equation is:

$$y[i] = -a_1y[i-1] - a_2y[i-2] + b_0x[i] + b_1x[i-1] + b_2x[i-2] \quad (6-6)$$

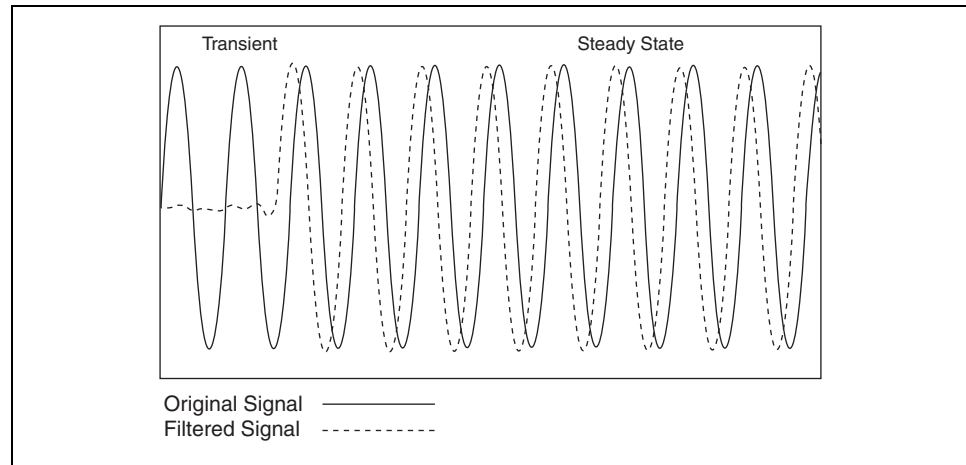
To calculate the current output (at the i^{th} instant) of the filter, you need to know the past two outputs (at the $(i-1)^{\text{st}}$ and the $(i-2)^{\text{nd}}$ time instants) and the current input (at the i^{th} time instant) and past two inputs (at the $(i-1)^{\text{st}}$ and $(i-2)^{\text{nd}}$ time instants).

Suppose that you start the filtering process by taking the first sample of the input data. At this time instant, you do not have the previous inputs ($x[i-1]$ and $x[i-2]$) or previous outputs ($y[i-1]$ and $y[i-2]$). By default, these values are assumed to be zero. When you get the second data sample, you have the previous input ($x[i-1]$) and the previous output ($y[i-1]$) that you calculated from the first sample but not $x[i-2]$ and $y[i-2]$. Again, by default, these are assumed to be zero. After you start processing the third input data sample, all the terms on the right hand side of Equation 6-6 have the previously calculated values. A certain amount of delay occurs before which calculated values are available for all the terms on the right hand side (RHS) of the difference equation that describes the filter. The output of the filter during this time interval is a transient and is known as the transient response. For lowpass and highpass filters, the duration of the transient response, or delay, is equal to the order of the filter. For bandpass and bandstop filters, this delay is $2 \times$ order.

IIR filters contain the following properties.

- Negative indices that result from Equation 6-4 are assumed to be zero the first time you call the VI.
- Because the initial filter state is assumed to be zero (negative indices), a transient proportional to the filter order occurs before the filter reaches a steady state. The duration of the transient response, or delay, for lowpass and highpass filters is equal to the order of the filter.
- The duration of the transient response for bandpass and bandstop filters is twice the filter order.

Each time a Filter VI is called, this transient appears at the output, as illustrated in the following illustration. You can eliminate this transient response on successive calls by enabling the state memory of the VI. To enable state memory, set the **init/cont** parameter of the VI to TRUE (continuous filtering). You will learn how to do this in a later exercise.



- The number of elements in the filtered sequence equals the number of elements in the input sequence.
- The filter retains the internal filter state values when the filtering completes.



- b. Place the DAQmx Timing VI, located on the **Functions»All Functions»NI Measurements»DAQmx - Data Acquisition** palette, on the block diagram. This VI configures the timing for the task. Right-click the sample mode input and select **Create»Constant** from the shortcut menu. Enter `Continuous Samples` in the constant. Right-click the **samples per channel** input and select **Create»Control** from the shortcut menu. This is the buffer value. Right-click the **rate** terminal and select **Create»Control** from the shortcut menu.



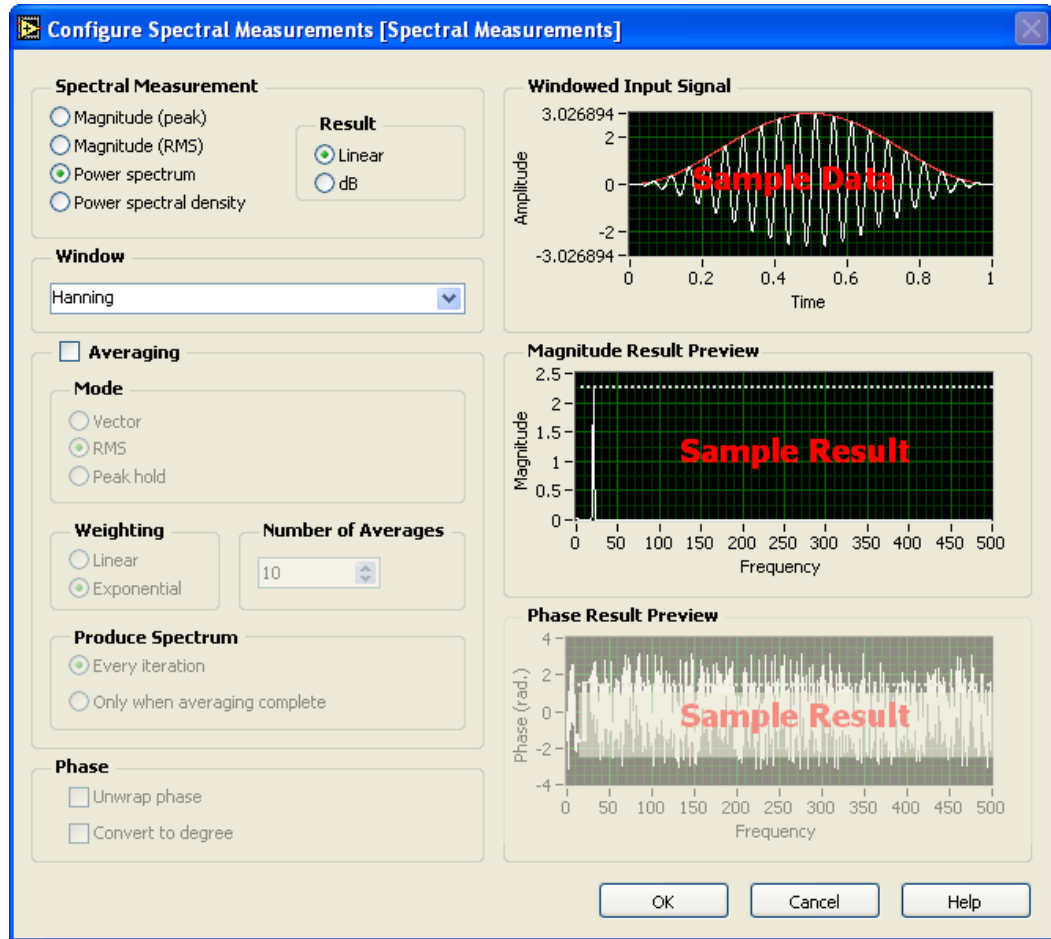
- c. Place the DAQmx Start VI, located on the **Functions»All Functions»NI Measurements»DAQmx - Data Acquisition** palette, on the block diagram. This VI starts the continuous buffered analog input operation.



- d. Place the DAQmx Read VI, located on the **Functions»All Functions»NI Measurements»DAQmx - Data Acquisition** palette, on the block diagram. This VI Reads data from the buffer allocated by the DAQmx Timing VI. Select the **Analog»Single Channel»Multiple Samples»Waveform** instance from the pull-down menu.



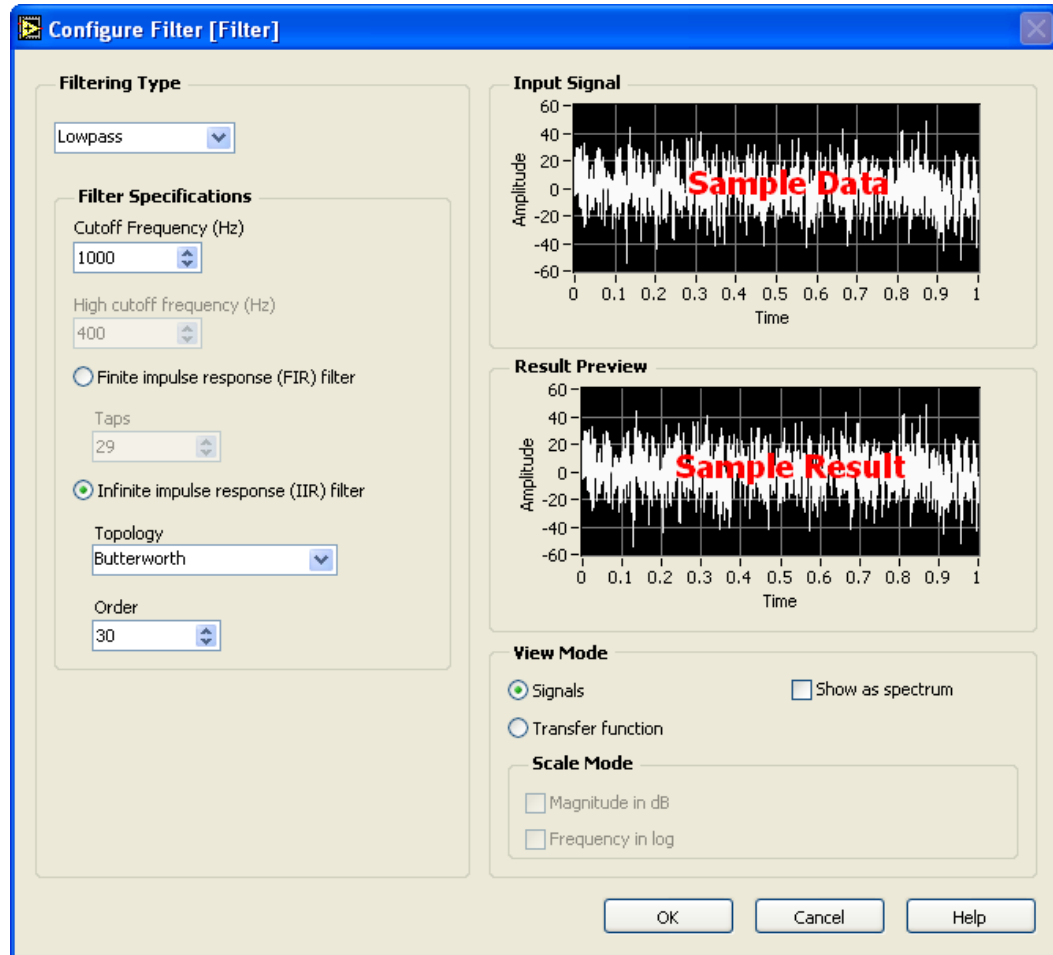
- e. Place the Spectral Measurements Express VI, located on the **Functions»Signal Analysis** palette, on the block diagram. This VI computes the power spectrum of an input signal. In the **Configure Spectral Measurements** dialog box that displays, set the options as shown in the following figure.



- Configure both instances of the Spectral Measurements Express VI with these settings.
- Click the **OK** button to close the dialog box.

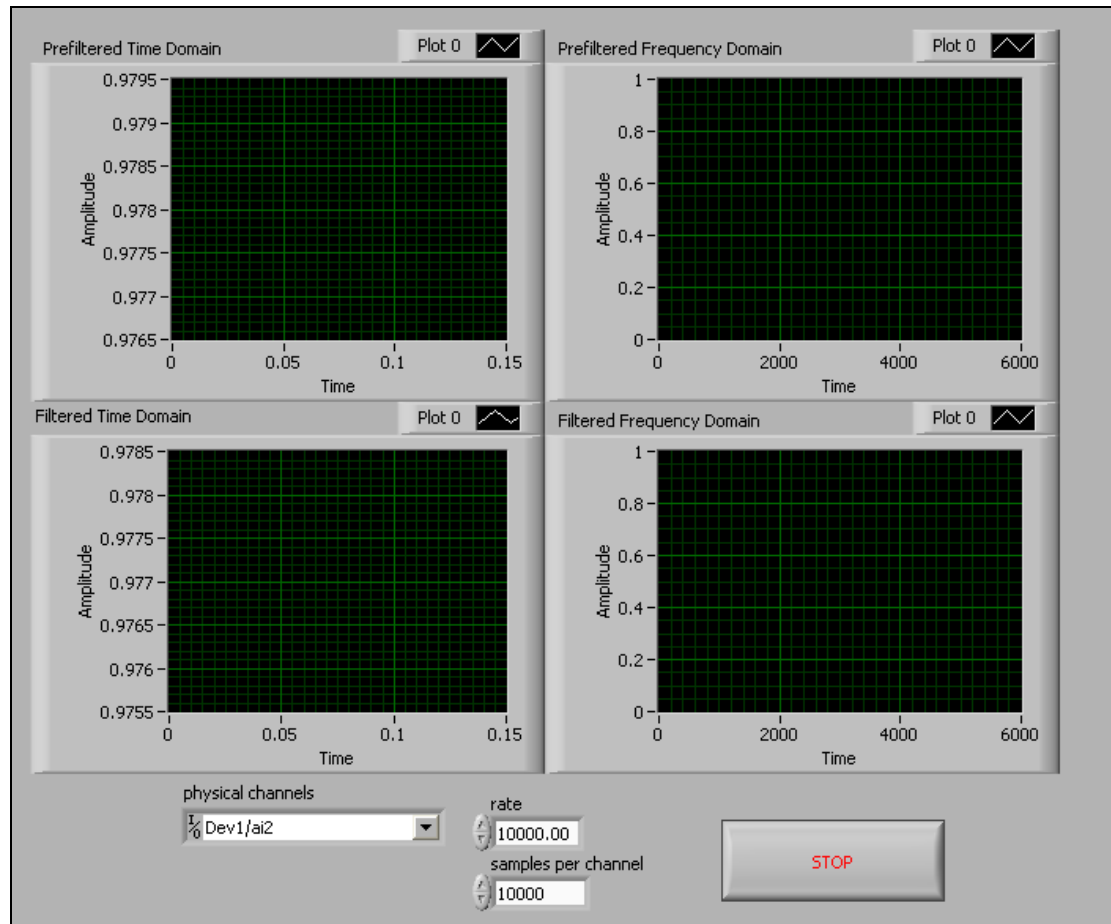


- f. Place the Filter Express VI, located on the **Functions»Signal Analysis** palette, on the block diagram. This Express VI processes signals through filters and windows. In the **Configure Filter** dialog box that displays, set the options as shown in the following figure.



- g. Place the DAQmx Stop Task VI, located on the **Functions»All Functions»NI Measurements»DAQmx - Data Acquisition** palette, on the block diagram. This VI stops the given task.
- h. Place the Unbundle by Name function, located on the **Functions»All Functions»Cluster** palette, on the block diagram. This function returns the Boolean **status** value from the error cluster.
- i. Place the Merge Errors VI, located on the **Functions»All Functions»Time & Dialog** palette, on the block diagram. This VI merges error clusters from different functions.
- j. Place the Simple Error Handler VI, located on the **Functions»All Functions»Time & Dialog** palette, on the block diagram. In the event of an error, this VI displays a dialog box with information regarding the error and where it occurred.

- Switch to the front panel and arrange the front panel as shown in the following figure.



- Save the VI as `IIR DAQ Filter.vi` in the `C:\Exercises\LabVIEW DAQ` directory.
- On the front panel of the VI, set the **physical channels** to `DevX/ai2`, where `X` corresponds to the device number of your DAQ device. Set **rate** to 10000 and **samples per channel** to 10000.
- Wire the DAQ Signal Accessory analog out 1 to analog in 2. These settings tell the VI to acquire data from channel 2 at a rate of 10 kHz, implying that the Nyquist frequency is 5 kHz.
- Select **Help»Find Examples** to open the NI Example Finder and browse by task to **Hardware Input and Output»DAQmx»Analog Generation»Voltage** and open the Cont Gen Voltage Wfm-Int Clk-Variable Rate VI. Run the VI and generate a 100 Hz sine wave on analog output channel 1.

10. Run the IIR DAQ Filter VI. Notice the four waveforms that are graphed.

In the Prefiltered Time Domain waveform graph, the acquired data is a very discrete sine wave because digital data generates a discrete analog waveform. The generated sine wave contains very high frequency components beyond the fundamental. These high frequency components appear in the Prefiltered Frequency Domain waveform graph. The lowpass Digital IIR filter attenuates these high frequency components with the almost perfect sine wave in the Filtered Time Domain waveform graph and Filtered Frequency Domain graph. Always be careful with aliasing. A digital filter will not remove aliasing.

11. Aliasing is a problem with digital filtering. Slowly increase the frequency the example VI generates. As you increase the frequency beyond 500 Hz, the signal is attenuated in the filtered waveform graphs. However, the signal is never completely attenuated. The pseudo-analog waveform that the example VI generates contains a lot of high frequency information. Because there are no hardware anti-aliasing filters, the high frequency components are aliased between 0 and 5 kHz. The digital IIR filter attenuates these high frequency aliased components but never completely eliminates them. This emphasizes the importance of hardware anti-aliasing filters.
12. Change the frequency that the example VI generates to 500 Hz and change the Lower FC to 400 Hz while keeping a Butterworth filter topology and filter order of 5. To do this, stop the VI and double-click the Filter Express VI. The input signal is not completely attenuated. The Butterworth filter transition region gets steeper as the order increases. Increase the order, and the filter begins to attenuate more of the signal. Increase the order until you are satisfied that the filter has attenuated the signal enough. A typical filter order is about 1 to 20, with 30 being an upper limit. With a higher order filter, the computer needs more processing time, and the chance increases of introducing floating point errors into the output.
13. Experiment with different filter topologies, filter orders, and cutoff frequencies.
14. Stop and close the VI when you finish.

End of Exercise 6-3

0. Finite Impulse Response Filters

Because the output of an IIR filter depends on the previous outputs and the current and previous inputs, the IIR filter has infinite memory, resulting in an impulse response of infinite length.

Finite Impulse Response (FIR) filters are digital filters with a finite impulse response. FIR filters are also known as nonrecursive filters, convolution filters, or moving-average (MA) filters because you can express the output of a FIR filter as a finite convolution

$$y_i = \sum_{k=0}^{n-1} h_k x_{i-k}$$

where x represents the input sequence to be filtered, y represents the output filtered sequence, and h represents the FIR filter coefficients.

The output of an FIR filter depends only on the current and past inputs. Because it does not depend on the past outputs, its impulse response decays to zero in a finite amount of time. The output of a general FIR filter is given by

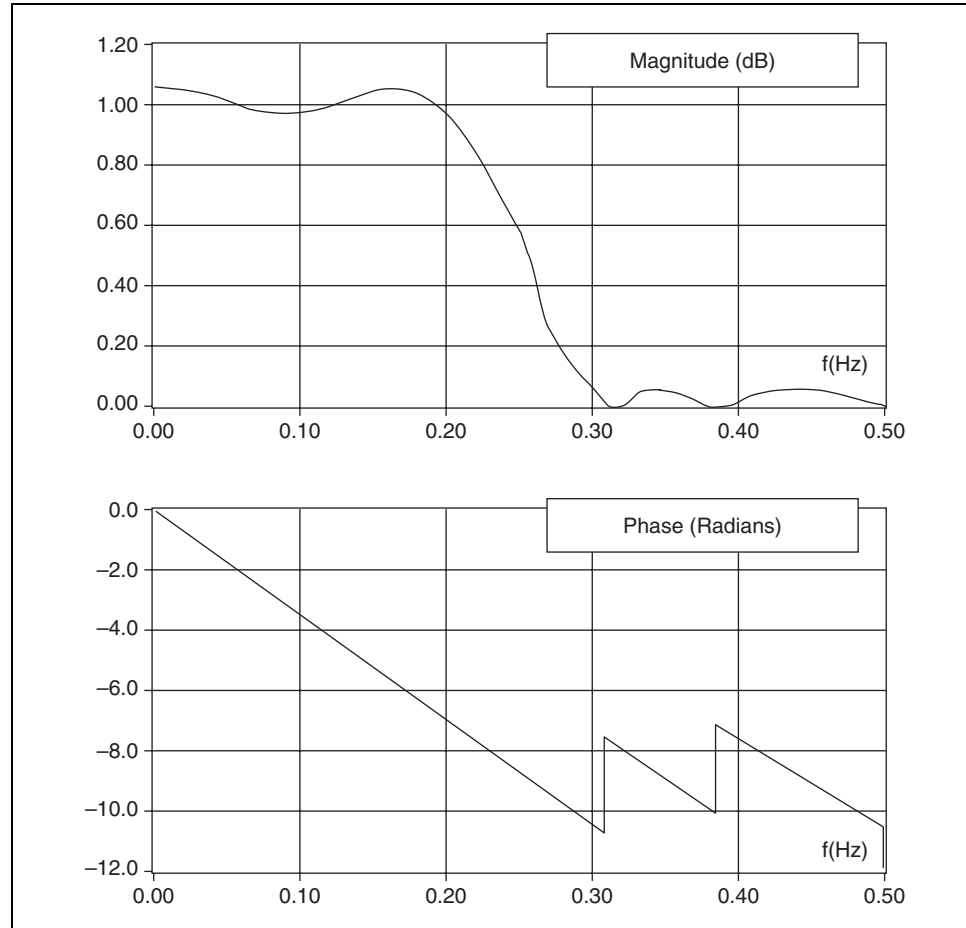
$$y[i] = b_0x[i] + b_1x[i-1] + b_2x[i-2] + \dots + b_{N-1}x[i-(M-1)]$$

where M is the number of taps of the filter, and b_0, b_1, \dots, b_{M-1} are its coefficients.

FIR filters have some important characteristics:

- They can achieve linear phase response and pass a signal without phase distortion.
- They are always stable. During filter design or development, you do not need to worry about stability concerns.
- FIR filters are simpler and easier to implement than IIR filters.

The following illustration plots a typical magnitude and phase response of FIR filters versus normalized frequency. The discontinuities in the phase response arise from the discontinuities introduced when you compute the magnitude response using the absolute value. The discontinuities in phase are on the order of π . The phase, however, is clearly linear.



The simplest method for designing linear-phase FIR filters is the window design method. To design a FIR filter by windowing, start with an ideal frequency response, calculate its impulse response, and then truncate the impulse response to produce a finite number of coefficients. The truncation of the ideal impulse response results in the effect known as the Gibbs phenomenon—oscillatory behavior near abrupt transitions (cutoff frequencies) in the FIR filter frequency response.

You can reduce the effects of the Gibbs phenomenon by smoothing the truncation of the ideal impulse response using a smoothing window. By tapering the FIR coefficients at each end, you can diminish the height of the side lobes in the frequency response. The disadvantage of this method is that the main lobe widens, resulting in a wider transition region at the cutoff frequencies.

The selection of a window function is similar to the choice between Chebyshev and Butterworth IIR filters in that it is a trade off between side lobes near the cutoff frequencies and the width of the transition region.

Summary

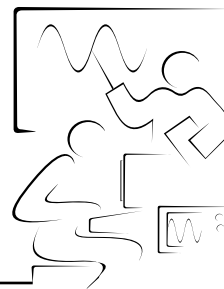
- Fast calculation of the Discrete Fourier Transform (DFT) is possible by using an algorithm known as the fast Fourier transform (FFT). You can use this algorithm when the number of signal samples is a power of two.
- The power spectrum can be calculated from the DFT/FFT by squaring the magnitude of the individual frequency components. The Power Spectrum VI in the advanced analysis library does this automatically for you. The Power Spectrum VI units of the output are V_{rms}^2 . The power spectrum does not provide any phase information.
- The DFT, FFT, and power spectrum are useful for measuring the frequency content of stationary or transient signals. The FFT provides the average frequency content of the signal over the entire time that the signal was acquired. Use the FFT mostly for stationary signal analysis when the signal is not significantly changing in frequency content over the time that the signal is acquired or when you want only the average energy at each frequency line.
- To reduce the spectral leakage, the finite time waveform is multiplied by a window function.
- Windows can be used to separate two sine waves that have widely different amplitudes but are very close in frequency.
- For practical filters, the gain in the passband may not always be equal to 1, the attenuation in the stopband may not always be $-\infty$, and there exists a transition region of finite width.
- The width of the transition region depends on the filter order, and the width decreases with increasing order.
- The output of FIR filters depends only on the current and past input values, but the output of IIR filters depends on the current and past input values and the past output values.
- IIR filters can be classified according to the presence of ripples in the passband and/or the stopband.
- Because of the dependence of its output on past outputs, a transient appears at the output of an IIR filter each time the VI is called. This transient can be eliminated after the first call to the VI by setting its **init/cont** control to TRUE.
- Use the Spectral Measurements Express VI and Filter Express VI to easily configure spectral calculations and filtering options for a given input signal.

Notes

Notes

Lesson 7

Analog Output



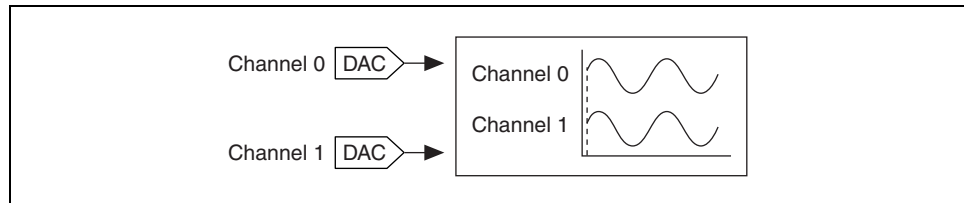
In this lesson, you will learn how to generate an analog signal. Much of what you learned about analog input has a parallel in analog output.

You Will Learn:

- A. Architecture of analog output
- B. About the Analog Output VIs
- C. How to perform single point analog output
- D. How to perform continuous analog output
- E. How to perform buffered analog output
- F. How to perform continuous buffered analog output
- G. How to perform triggered analog output

A. Analog Output Architecture

Most E Series devices have a digital-to-analog converter (DAC) for each analog output channel. All DACs are updated at the same time, so only one clock signal is needed. The analog output clock is the update clock. The output of the analog output channels is synchronized, similar to the way analog input channels are synchronized during simultaneous sampling.



Analog Output Considerations

The DAC has a range that is determined by a reference voltage. The reference voltage can be an internal signal or an external signal. The internal reference voltage is a +10 V signal. You can set the range of the DAC as bipolar or unipolar.

Bipolar

A bipolar signal has a range that includes positive and negative values. If you set the device in bipolar mode, the range of the DAC is determined as follows:

- Maximum voltage = $+V_{\text{ref}}$
- Minimum voltage = $-V_{\text{ref}}$

For example, if you are using the +10 V internal reference voltage, the range of the DAC would be set at -10 to $+10$ V. However, if the signal only goes from -5 to $+5$ V, you are not maximizing the resolution of the DAC. To maximize the resolution, you could provide an external reference voltage with a value of $+5$ V. Now the range of the DAC is -5 to $+5$ V, the same as the signal, and you are using the full resolution of the DAC to generate the signal.

Unipolar

A unipolar signal has a range that includes only positive values. If you set the device in unipolar mode, the range of the DAC is determined as follows:

- Maximum voltage = $+V_{\text{ref}}$
- Minimum voltage = 0 V

For example, if you use the +10 V internal reference voltage, the range of the DAC is set at 0 V to +10 V. If the signal is only 0 V to +5 V, you are not maximizing the resolution of the DAC. To maximize the resolution, you can provide an external reference voltage with a value of +5 V. The range of the DAC is 0 V to +5 V, the same range as the signal, so you will be using the full resolution of the DAC to generate the signal.

B. Using the DAQmx Write VI

The DAQmx Write VI located on the **DAQmx - Data Acquisition** palette writes samples to the task or channels you specify. The instances of this polymorphic VI specify the format of the samples to write, whether to write one or multiple samples, and whether to write to one or multiple channels. This lesson describes the analog output instance of the DAQmx Write VI. Use the pull-down menu to select the instance of this VI.

There are four selection windows used to determine the instance of the Write VI. The first selection window allows you to choose the type of output:

- Analog
- Digital
- Raw data

The second selection window determines the number of channels to write to or if the data is of the unscaled type. The third selection window allows you to choose to output either a single sample or multiple samples. For a single sample output, the fourth selection window will allow you to select the data to be written as either a waveform or double value. For a multiple sample output, the fourth selection window will allow you to select the data to be written as a waveform or an array of double values.

For a single sample output, the **auto start** terminal is by default set to true. This is because the task state model can be entirely implicitly controlled when outputting a single output. However, for a multiple sample output, the **auto start** terminal is by default set to false. This is because additional timing must be configured when outputting multiple samples with the DAQmx Timing VI and the DAQmx Start Task VI and DAQmx Stop Task VI must also be used. For more information about the DAQmx Task State Model, refer to Lesson 2, *Data Acquisition Hardware and Software*.

Single Sample Generation

If the signal level is more important than the generation rate, output just a single sample. Generate one sample at a time when you need to generate a constant, or DC, signal. To control when your device generates a signal, you can use either software or hardware timing.

- **Software-timed**—The rate at which samples are generated is determined by the software application and the operating system, not the DAQ device. Because the generation is entirely dependent on your operating system's resources, any interruption in the system can possibly affect the generation.
- **Hardware-timed**—A TTL signal, such as a clock on the DAQ device, controls the rate of generation. A hardware-time generation can run at a much faster rate than a software-time generation and maintain a higher value of accuracy. Not all devices support hardware timing. Consult your device's documentation to verify if it does or does not support hardware timing.

Setting the Timing for an Analog Output Generation

To inform DAQmx to use either hardware or software timing, use the DAQmx Timing VI and/or the DAQmx Sample Timing Type property node. By selecting the **Sample Clock** instance of the Timing VI or setting the Sample Timing Type property node to Sample Clock, you are telling DAQmx to use the Sample Clock from your DAQ device to control the generation. For a software-time generation, set the Sample Timing Type property node to **On Demand**. If you do not specify the timing type with the DAQmx Timing VI or the DAQmx Sample Timing Type property node, you will be using a software-timed generation.

In addition, the DAQmx Timing VI contains a **Use Waveform** instance. This instance uses the **dt** component of the waveform input to determine the sample clock rate. **dt** is the time in seconds between samples. This will establish hardware-timing for the analog generation. The Use Waveform instance of the DAQmx Timing VI does not actually output the waveform values, it only uses the waveform to configure the timing. Wire the waveform to the DAQmx Write VI to produce the samples.

Exercise 7-1 Single-Point Generation

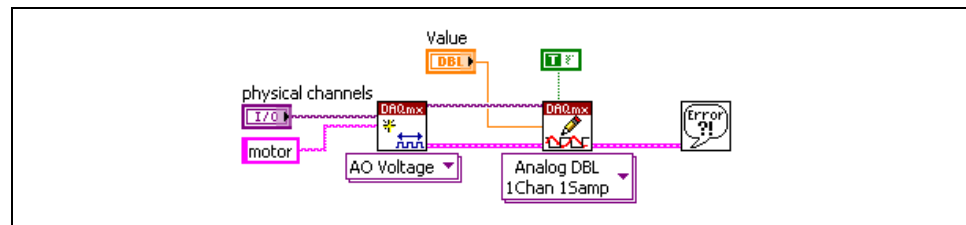
Objective: To build a VI that generates a + 5 V signal.

In this exercise, you use an analog output channel on the DAQ device to power a servomotor.



Note The speed of a servomotor is proportional to the voltage with which it is driven. As the input voltage increases, the speed of the motor increases.

1. On the DAQ Signal Accessory, connect analog out 0 to analog in 1.
2. Open a blank VI and build the following block diagram. To create the controls and constants, right-click the inputs and select **Create»Control** or **Create»Constant** from the shortcut menu.



- a. Place the DAQmx Create Virtual Channel VI, located on the **Functions»All Functions»NI Measurements»DAQmx - Data Acquisition** palette, on the block diagram. This VI creates a virtual channel based on the input terminal settings. Select the **Analog Output»Voltage** channel type from the pull-down menu. Name this channel `motor`.



- b. Place the DAQmx Write VI, located on the **Functions»All Functions»NI Measurements»DAQmx - Data Acquisition** palette, on the block diagram. This polymorphic VI outputs a voltage based on the type you specify in the pull-down menu. Select the **Analog»Single Channel»Single Sample»DBL** instance from the pull-down menu. Since you chose a single sample, the default value for **auto start** is True. In this instance of the DAQmx Write VI, you are not required to use the Start/Stop Task VIs. However, you will add the Start/Stop Task VIs to this VI in a later exercise.



- c. Place the Simple Error Handler VI, located on the **Functions»All Functions»Time & Dialog** palette, on the block diagram. In the event of an error, this VI displays a dialog box with information regarding the error and where it occurred.

3. On the front panel, set the controls with the following values:
 - **Physical channel:** DevX/ao0, where *X* corresponds to the device number of your DAQ device
 - **Value:** 5
4. Save the VI as `Servo Fan.vi` in the `C:\Exercises\LabVIEW DAQ` directory.
5. Run the VI. The DAQ device produces +5 V on analog out 0 on the DAQ Signal Accessory. Since you did not explicitly set the timing to use the Sample Clock, the generation is software-timed.
6. Open the Voltmeter VI located in the `C:\Exercises\LabVIEW DAQ` directory.
7. Set the control on the front panel with the following value:

Physical channel: DevX/ai1, where *X* corresponds to the device number of your DAQ device.
8. Run the Voltmeter VI. You should see +5 V appear on the meter.
9. Close the VI when you finish.

End of Exercise 7-1

Exercise 7-2 Continuous Single-Point Generation

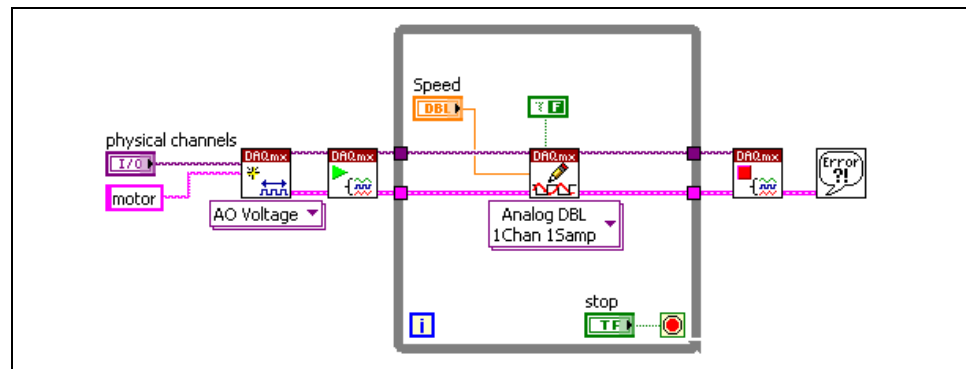
Objective: To build a VI that generates a variable voltage.

In this exercise, you modify the Servo Fan VI you created in Exercise 7-1 to control the speed of the fan. To add this capability, you must be able to continuously update the analog output channel on the DAQ device.

1. On the DAQ Signal Accessory, connect analog out 0 to analog in 1.
2. Open the Servo Fan VI located in the C:\Exercises\LabVIEW DAQ directory.
3. Select **File»Save As** and save the VI as Variable Servo Fan.vi in the C:\Exercises\LabVIEW DAQ directory.
4. On the front panel, replace the numeric control **Value** with a horizontal pointer slide located on the **Controls»Numeric Controls** palette by right-clicking the numeric control and selecting **Replace** from the shortcut menu. Select the horizontal pointer slide from the **Numeric Controls** palette. Re-label the control **Speed**.

Block Diagram

5. Modify the block diagram as shown in the following figure.

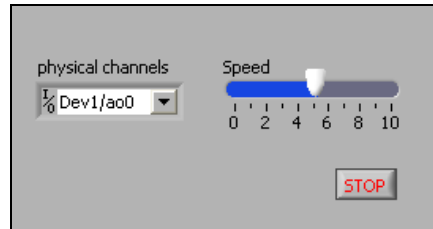


- a. Place the DAQmx Start Task VI, located on the **Functions»All Functions»NI Measurements»DAQmx - Data Acquisition** palette, on the block diagram. This VI starts a given task. Because you are using the Start Task VI, you must change the **auto start** input on the DAQmx Write VI to False.



- b. Place the DAQmx Stop Task VI, located on the **Functions»All Functions»NI Measurements»DAQmx - Data Acquisition** palette, on the block diagram. This VI stops a given task.

6. Modify the front panel as shown in the following figure.



7. Save the VI.
8. Open the Continuous Acquire with MIO VI located in the C:\Exercises\LabVIEW DAQ directory. Set the front panel controls as follows:
- **Physical Channels:** DevX/ai1, where X corresponds to the device number of your DAQ device
 - **Samples per channel:** 1000
 - **Rate:** 1000
9. Run the Continuous Acquire with MIO VI.
10. On the front panel of the Variable Servo Fan VI, set the controls as follows:
- Physical Channels:** DevX/ai1, where X corresponds to the device number of your DAQ device.
11. Run the Variable Servo Fan VI. Adjust the **Speed** control and observe the voltage that is produced.
12. Close the VIs when you finish.

End of Exercise 7-2

C. Multiple-Point (Buffered) AO VIs

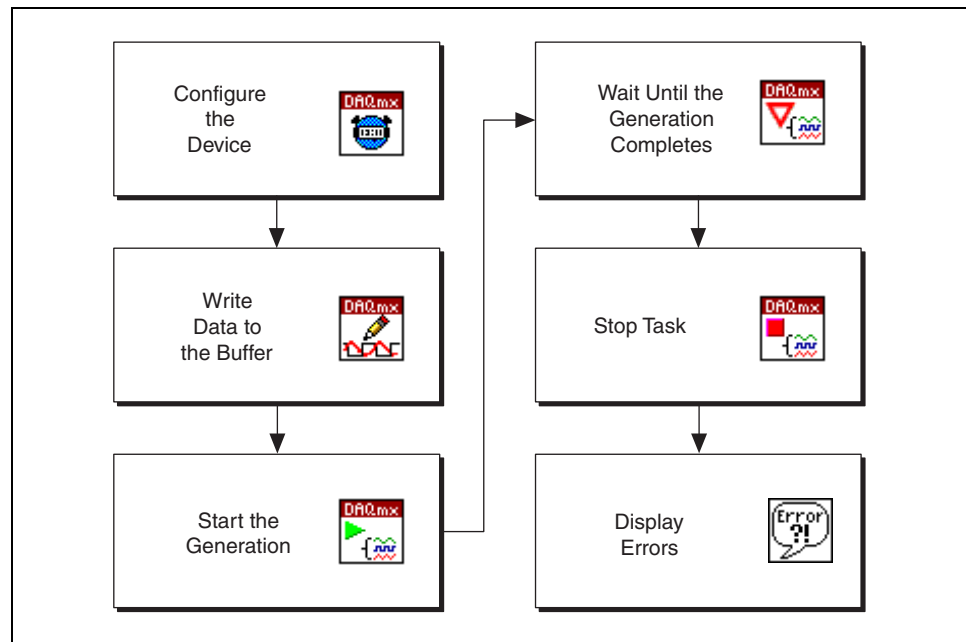
To generate multiple analog output samples, configure the pull-down menu of the DAQmx Write VI for multiple samples. Use a multiple sample generation if you want to generate a time-varying signal, such as an AC sine wave. Multiple point generation also is known as buffered analog output. Buffered analog output can be either finite or continuous. However, both buffering methods involve the following two main steps:

1. Write samples into the buffer. The points are taken from LabVIEW and placed in an intermediate memory buffer before they are sent to the device. A buffered generation is similar to sending an entire email at once instead of sending it one word at a time.
2. Transfer samples from the buffer to the DAQ device. The rate at which the samples are transferred depends on the timing that you specify. As with single sample generation, you can use either a software or hardware-timed generation.

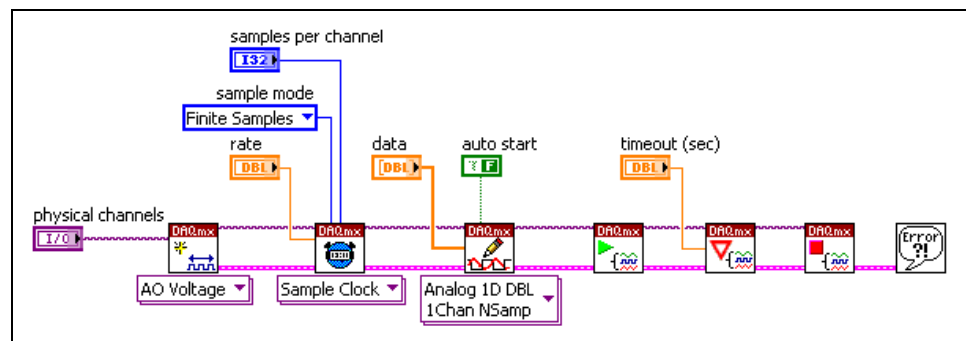
In a hardware-timed generation, a hardware signal called the update clock controls the rate of generation. A hardware clock can run much faster than a software loop, so you can generate a wider range of signal frequencies and shapes. A hardware clock also is more accurate than a software loop. A software loop rate can be interfered with by a variety of actions such as the opening of another program on the computer.

D. Finite Buffered Generation

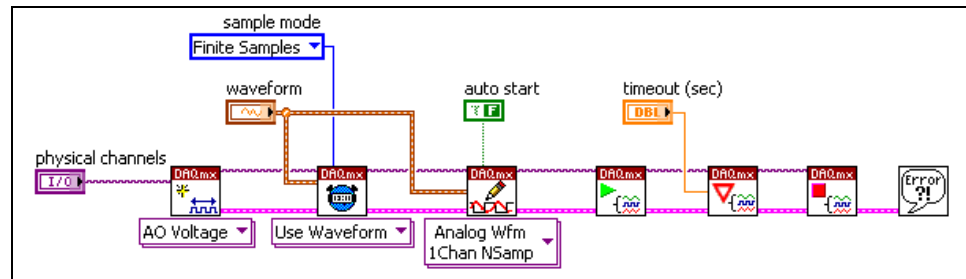
The following illustration shows a flowchart of a buffered generation.



The following figure shows an example of a typical finite buffered generation using the Sample Clock and an array of doubles for the output data.



You also can use the waveform data type to determine the timing and sample data, as seen in the following figure.



There are few differences between the two types of finite buffered generation—using an array of doubles with the Sample Clock and using a waveform data type to set the Sample Clock and samples. The DAQmx Timing VI instance changes, as well as the data that is wired to the **data** terminal of the DAQmx Write VI. Both types follow the same general structure that we will now describe.

The DAQmx Create Virtual Channel VI can be used to programmatically create an analog output virtual channel. If you have already created your virtual channel or task using the DAQ Assistant in MAX, you can skip this VI and wire the channel/task to the next VI, the DAQmx Timing VI.

The DAQmx Timing VI has two instances that we can use for analog output—Sample Clock and Use Waveform. Since we are generating a finite number of samples, set the **sample mode** to Finite Samples for both instances. When using the Sample Clock, we also specify the generation **rate** and **number of samples**. The number of samples value determines the buffer size. For the Use Waveform instance, simply wire the waveform data type to the **waveform** terminal. This instance of the VI will determine the sample clock rate and number of samples (buffer size) based on the data contained in the waveform.

The DAQmx Write VI actually sends the data to the PC buffer. You can select to output either a waveform data type or an array of doubles. For the Use Waveform instance of the DAQmx Timing VI, select to output a waveform type from the pull-down menu of the Write VI. Wire the same waveform you used to set the timing to the **data** terminal of the Write VI. When using the built-in Sample Clock for timing, select to output an array of doubles from the pull-down menu of the Write VI. Then, wire the array of doubles you wish to generate to the **data** terminal of the Write VI.

For multiple samples, the **auto start** parameter defaults to a False value. Since we will be explicitly start the task, waiting for it to finish, and then stopping the task, we want to leave the auto start value to False.

The DAQmx Start VI begins the generation. The DAQmx Wait Until Done VI waits until the task has completed, or a timeout occurs. When either of these occur, control is then passed to the DAQmx Stop Task VI and the task is stopped. As is usual with the LabVIEW programming in this course, the error cluster is passed to each VI and if necessary, an error message is displayed.

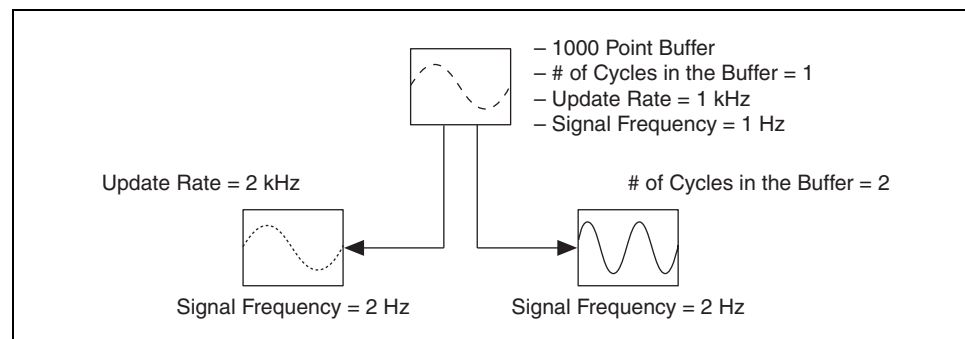
DAQmx Reset VI

In analog output, once you write a value to an analog output channel, the channel continues to output that voltage until a new value is written to the analog output channel or the device is reset by the DAQmx Reset VI (located on the **DAQmx - Data Acquisition»DAQmx Device Configuration** palette) or the device is powered off.

Assume you are writing a sine wave to an analog output channel and the last value in the buffer is seven. You generate the entire sine wave, and after the generation is complete, the analog output channel continues to generate a voltage of seven. Rather than resetting the device every time, it is easier to write a value of zero to the channel after the generation is complete. You can use the AO Write One Update VI located on the **Utility** palette to perform just such an operation.

Output Waveform Frequency

The frequency of the output waveform depends on the update rate and the number of cycles of the waveform present in the buffer, as illustrated in the following illustration.



The formula for calculating the frequency of the output signal is as follows:

$$\text{signal frequency} = [(\text{cycles}) \times (\text{update rate})] / (\text{points in the buffer})$$

The following example illustrates how the update rate and the number of cycles of the waveform in the buffer affect the signal frequency. Assume you have a 1,000 point buffer that holds one cycle of the waveform. If you generate the signal with an update rate of 1 kHz, the signal frequency is as follows:

$$[(1 \text{ cycle}) \times (1,000 \text{ points per second})]/(1,000 \text{ points}) = 1 \text{ Hz}$$

If you double the update rate and leave everything else the same, the signal frequency is as follows:

$$[(1 \text{ cycle}) \times (2,000 \text{ points per second})]/(1,000 \text{ points}) = 2 \text{ Hz}$$

If you double the number of cycles in the buffer and leave everything else the same, the signal frequency is as follows:

$$[(2 \text{ cycles}) \times (1,000 \text{ points per second})]/(1,000 \text{ point}) = 2\text{Hz}$$

Therefore, if you double the update rate or the number of cycles, you double the frequency of the output waveform.

Exercise 7-3 Buffered Generation

Objective: To build a VI that generates a waveform.



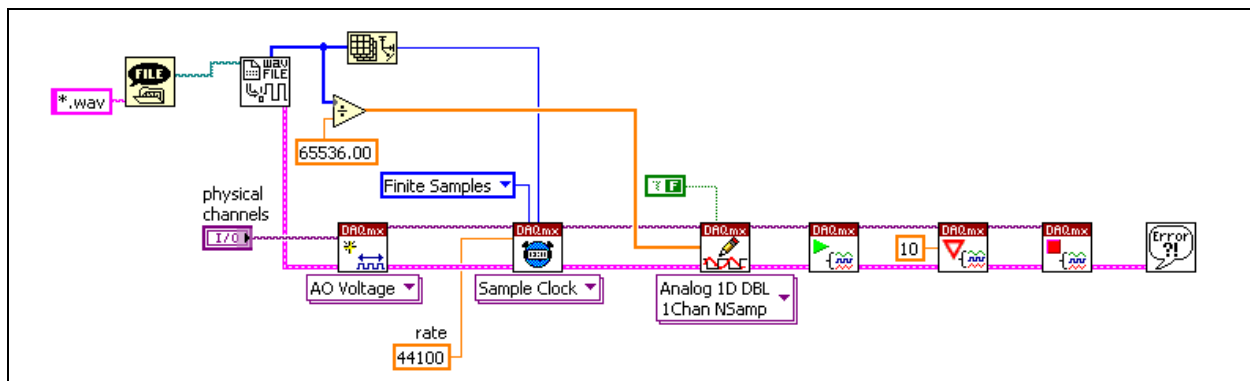
Note This exercise is designed for systems that have amplified speakers. If your system does not have amplified speakers, substitute Exercise 7-4.

In this exercise, you will build a VI that plays a WAV file from the DAQ device.

1. Connect one lead from the speaker input cable to analog out 0 on the DAQ Signal Accessory. Connect the other lead from the speaker input cable to GND on the DAQ Signal Accessory. You do not need to worry about polarity.
2. Open a blank VI.

Block Diagram

3. Build the block diagram shown in the following figure.



- a. Place the File Dialog function, located on the **Functions»All Functions»File I/O»Advanced File I/O Functions** palette, on the block diagram. This function displays a file dialog box. To locate only WAV files on the computer, right-click the **pattern** input and select **Create»Constant** from the shortcut menu. Enter *.wav to search only for WAV files.



- b. Place the Snd Read Wave File VI, located on the **Functions»All Functions»Graphics & Sound»Sound»Sound File** palette, on the block diagram. This VI retrieves a PC wave file (.wav), reads the header of the WAV file and returns the binary sound data located in the file. Select an output based on the type of WAV file that you read. For this case, use the **mono 16-bit** output.



c. Place the Divide function, located on the **Functions»Arithmetic & Comparison»Express Numeric** palette, on the block diagram. In this exercise, this function rescales the 16-bit ($2^{16} = 65,536$) data to a value between -1 and 1 .



d. Place the Array Size function, located on the **Functions»All Functions»Array** palette, on the block diagram. This function returns the number of samples that are located in the WAV file. You use this quantity to determine the samples per channel (buffer size) to allocate by the DAQmx Timing VI.



e. Place the DAQmx Create Virtual Channel VI, located on the **Functions»All Functions»NI Measurements»DAQmx - Data Acquisition** palette, on the block diagram. This VI creates a virtual channel for the physical channel that you specify. Select the **Analog Output»Voltage** type from the pull-down menu. Right-click the **physical channels** input and select **Create»Control** from the shortcut menu.



f. Place the DAQmx Timing VI, located on the **Functions»All Functions»NI Measurements»DAQmx - Data Acquisition** palette, on the block diagram. This VI configures the task timing. Right-click the **rate** input and select **Create»Constant** from the shortcut menu. The example WAV file is sampled at a rate of $44,100$ Hz, so enter 44100 for the rate.



g. Place the DAQmx Write VI, located on the **Functions»All Functions»NI Measurements»DAQmx - Data Acquisition** palette, on the block diagram. This VI writes data into the buffer for a finite, buffered analog output operation. Select the **Analog»Single Channel»Multiple Samples»1D DBL** instance from the pull-down menu.



h. Place the DAQmx Start Task VI, located on the **Functions»All Functions»NI Measurements»DAQmx - Data Acquisition** palette, on the block diagram. This VI starts the buffered analog output generation.



i. Place the DAQmx Wait Until Done VI, located on the **Functions»All Functions»NI Measurements»DAQmx - Data Acquisition** palette, on the block diagram. This VI waits until the analog output task completes before returning.



j. Place the DAQmx Stop Task VI, located on the **Functions»All Functions»NI Measurements»DAQmx - Data Acquisition** palette, on the block diagram. This VI stops the analog output task.



k. Place the Simple Error Handler VI, **Functions»All Functions»Time & Dialog** palette, on the block diagram. In the event of an error, this VI displays a dialog box with information regarding the error and where it occurred.

Front Panel

4. Set **physical channels** to DevX/ao0, where X corresponds to the device number of your DAQ device.
5. Save the VI as Tone Generation (7-3) .vi in the C:\Exercises\LabVIEW DAQ folder.
6. Adjust the volume on your speakers to a moderate level.
7. Run the VI. The VI prompts you to open a WAV file. Select the NISoundFile.wav located in the C:\Exercises\LabVIEW DAQ directory. You should hear the WAV file through the speakers.
8. Close the VI.

End of Exercise 7-3

Exercise 7-4 Buffered Generation (Optional)

Objective: To build a VI that generates a waveform.



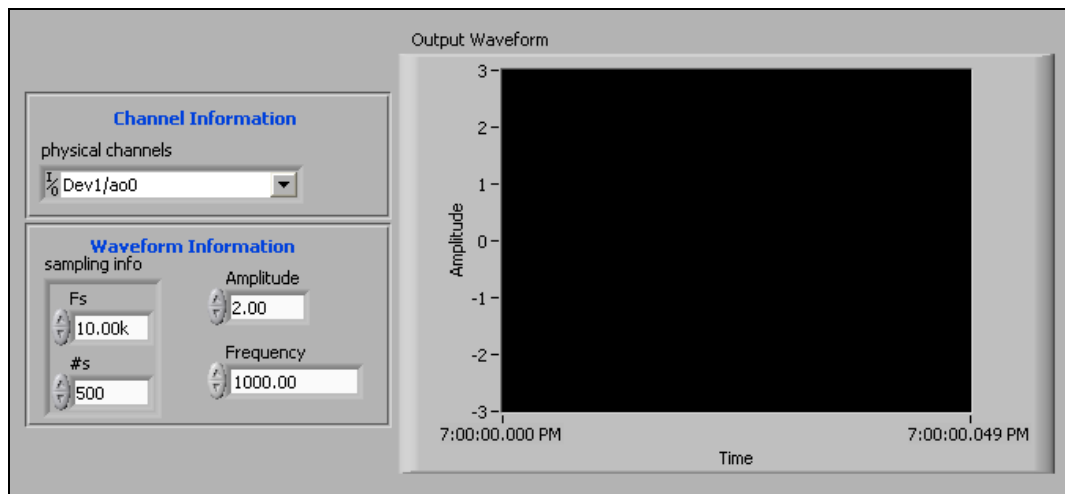
Note Substitute this exercise for Exercise 7-3 on systems that do not have amplified speakers.

In this exercise, you build a VI that produces an audible tone by producing a sine wave in the audible frequency range (20 Hz to 16 kHz) and using this sine wave to drive a speaker.

1. On the DAQ Signal Accessory, connect analog out 0 to analog in 1.

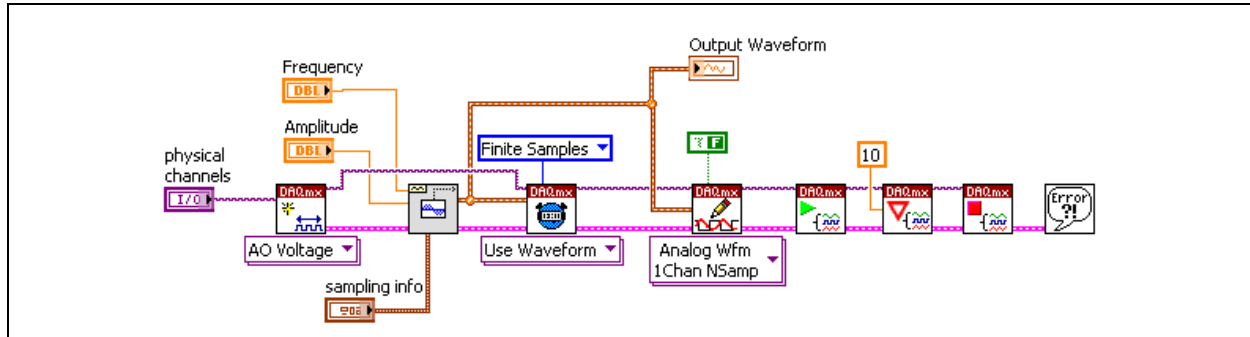
Front Panel

2. Open the Tone Generation (7-4) VI located in the C:\Exercises\LabVIEW DAQ directory. The following front panel displays.



Block Diagram

3. Build the following block diagram.



- a. Place the DAQmx Create Virtual Channel VI, located on the **Functions»All Functions»NI Measurements»DAQmx - Data Acquisition** palette, on the block diagram. This VI creates a virtual channel for the physical channel that you specify. Select the **Analog Output»Voltage** instance from the pull-down menu. Right-click the **physical channels** input and select **Create»Control** from the shortcut menu.



- b. Place the Sine Waveform VI, located on the **Functions»All Functions»Waveform»Waveform Generation** palette, on the block diagram. This VI generates a sine wave.



- c. Place the DAQmx Timing VI, located on the **Functions»All Functions»NI Measurements»DAQmx - Data Acquisition** palette, on the block diagram. This VI configures the task timing.



- d. Place the DAQmx Write VI, located on the **Functions»All Functions»NI Measurements»DAQmx - Data Acquisition** palette, on the block diagram. This VI writes data into the buffer for a finite, buffered analog output operation. Select the **Analog»Single Channel»Multiple Samples»Waveform** instance from the pull-down menu.



- e. Place the DAQmx Start Task VI, located on the **Functions»All Functions»NI Measurements»DAQmx - Data Acquisition** palette, on the block diagram. This VI starts the buffered analog output generation.



- f. Place the DAQmx Wait Until Done VI, located on the **Functions»All Functions»NI Measurements»DAQmx - Data Acquisition** palette, on the block diagram. This VI waits until the analog output task completes before returning data.



- g. Place the DAQmx Stop Task VI, located on the **Functions»All Functions»NI Measurements»DAQmx - Data Acquisition** palette, on the block diagram. This VI stops the analog output task.



- h. Place the Simple Error Handler VI, located on the **Functions»All Functions»Time & Dialog** palette, on the block diagram. In the

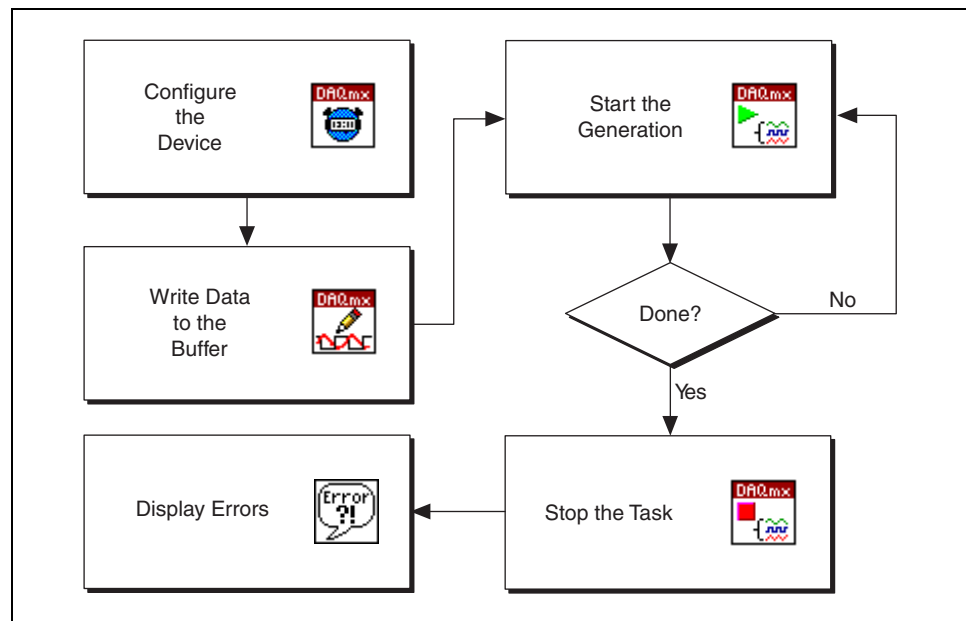
event of an error, this VI displays a dialog box with information regarding the error and where it occurred.

4. Save the VI.
5. Open the Continuous Acquire with MIO VI located in the C:\Exercises\LabVIEW DAQ folder. Use this VI to view the output of the Tone Generation (7-4) VI.
6. Set the controls on the front panel of the Continuous Acquire with MIO VI as follows:
 - **Physical Channel:** DevX/ai1, where X corresponds to the device number of your DAQ device
 - **Samples per Channel:** 5000
 - **Rate:** 10000
7. Run the Continuous Acquire with MIO VI.
8. On the front panel of the Tone Generation (7-4) VI, set the controls as follows:
 - **Physical Channel:** DevX/ao0, where X corresponds to the device number of your DAQ device
 - **Amplitude:** 1
 - **Frequency:** 10000
 - **Sampling Info**
 - **Fs:** 100000—Determines the sample rate of the waveform the Sine Waveform VI produces.
 - **#s:** 100000—Determines the number of samples in the waveform the Sine Waveform VI produces.
9. Run the Tone Generation (7-4) VI. A sine wave appears on the graph. If analog out 1 is connected to a speaker, you will hear an audible tone.
10. After the Tone Generation (7-4) VI stops, change the following controls on the front panel:
 - **Frequency:** 10
 - **Sampling Info**
 - **Fs:** 1000
 - **#s:** 1000
11. Run the Tone Generation (7-4) VI again.
12. Close the VIs.

End of Exercise 7-4

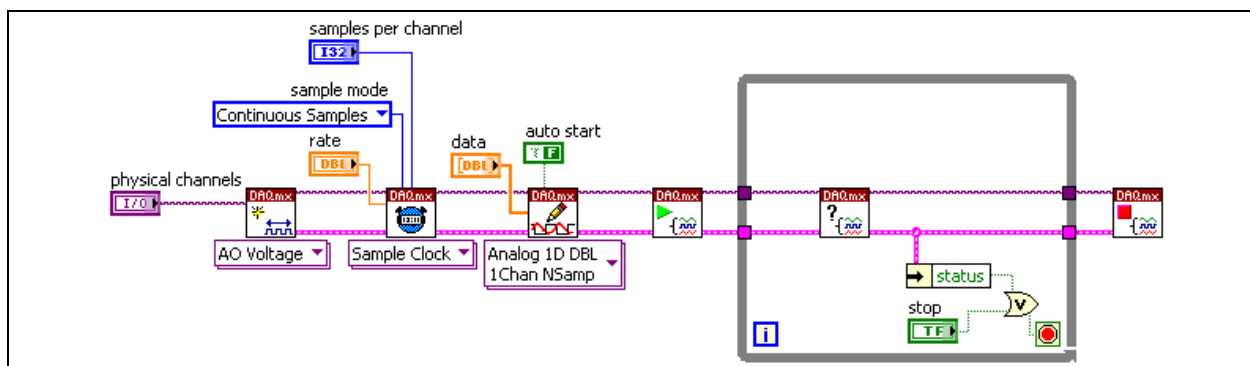
E. Continuous Buffered Generation

The main difference between a finite buffered generation and continuous buffered generation is the number of points that are generated. In a finite buffered generation, you generate the data in the buffer a finite number of times. In a continuous buffered generation, you can generate data indefinitely.



The following figure is similar to a buffered generation with the following differences:

- The DAQmx Timing VI is set to Continuous Samples sample mode.
- The DAQmx Is Task Done VI is used instead of the DAQmx Wait Until Done VI in a While Loop.



You start by configuring the virtual channel and timing settings with the Create Virtual Channel VI and Timing VI. Next you write to the buffer with the DAQmx Write VI and start the task with DAQmx Start Task VI.

The While Loop is used to poll the task to see if it has completed by using the DAQmx Is Task Done VI. The generation will stop when either the user clicks the stop button or an error occurs. The data in the buffer is being regenerated after it has all been processed. After the While Loop stops, the DAQmx Stop Task VI stops the tasks and any errors are then reported.



Note You can write new data to the buffer with each loop iteration. You must place a Write VI in the while loop and wire the new data to the data terminal. Using a Write property node, select the Regeneration Mode property and set it to not allow regeneration. You need to make sure you generate new data fast enough to prevent the buffer from regenerating old data. This situation is similar to performing a continuous buffered acquisition. You must make sure you read the data out of the buffer fast enough to prevent the data from being overwritten.

Exercise 7-5 Continuous Buffered Generation

Objective: To build a VI that continuously generates a waveform.



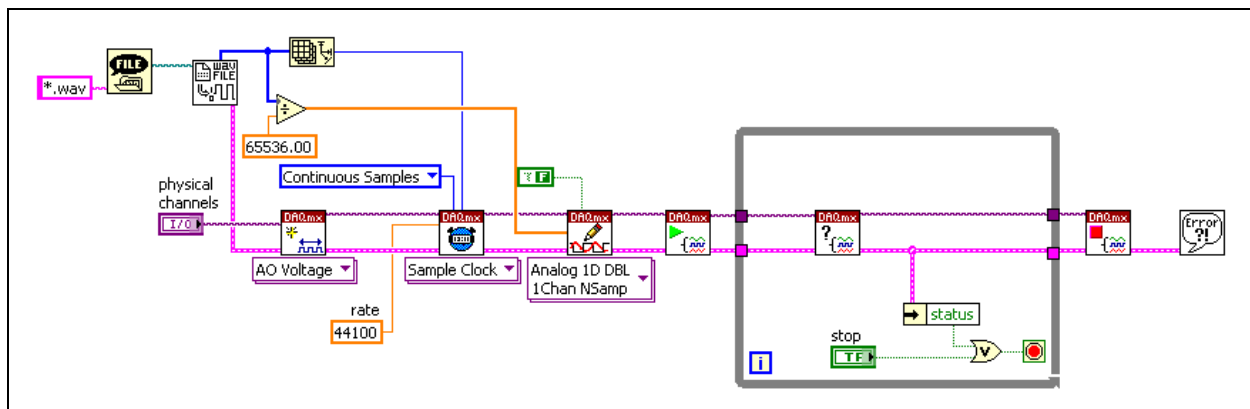
Note This exercise is designed for systems that have amplified speakers. If your system does not have amplified speakers, substitute Exercise 7-6.

In this exercise, you will build a VI that continuously plays a WAV file from the DAQ device.

1. Connect one lead from the speaker input cable to analog out 0 on the DAQ Signal Accessory. Connect the other lead from the speaker input cable to GND on the DAQ Signal Accessory. You do not need to worry about polarity.
2. Open the Tone Generation (7-3) VI that you completed in Exercise 7-3.
3. Save the VI as Continuous Tone Generation (7-5) .vi in the C:\Exercises\LabVIEW DAQ directory.

Block Diagram

4. Modify the block diagram as shown in the following figure.



- a. Place the DAQmx Is Task Done VI, located on the **Functions»All Functions»NI Measurements»DAQmx - Data Acquisition»DAQmx Advanced Task Options** palette, on the block diagram. This VI queries the status of a task and returns whether it completed execution. Place this VI in the While Loop to monitor the state of the task and wait for an error to occur or for the user to click the **STOP** button.
5. On the front panel, set the **physical channel** control to DevX/ao0, where X corresponds to the device number of your DAQ device.

6. Adjust the volume on your speakers to a moderate level.
7. Run the VI. The VI prompts you to open a WAV file. Select the `NISoundFile.wav` located in the `C:\Exercises\LabVIEW DAQ` directory.
8. You should hear the WAV file through the speakers. Click the **STOP** button to stop the VI.
9. Close the VI.

End of Exercise 7-5

Exercise 7-6 Continuous Buffered Generation (Optional)

Objective: To build a VI that continuously generates a waveform that simulates a siren.



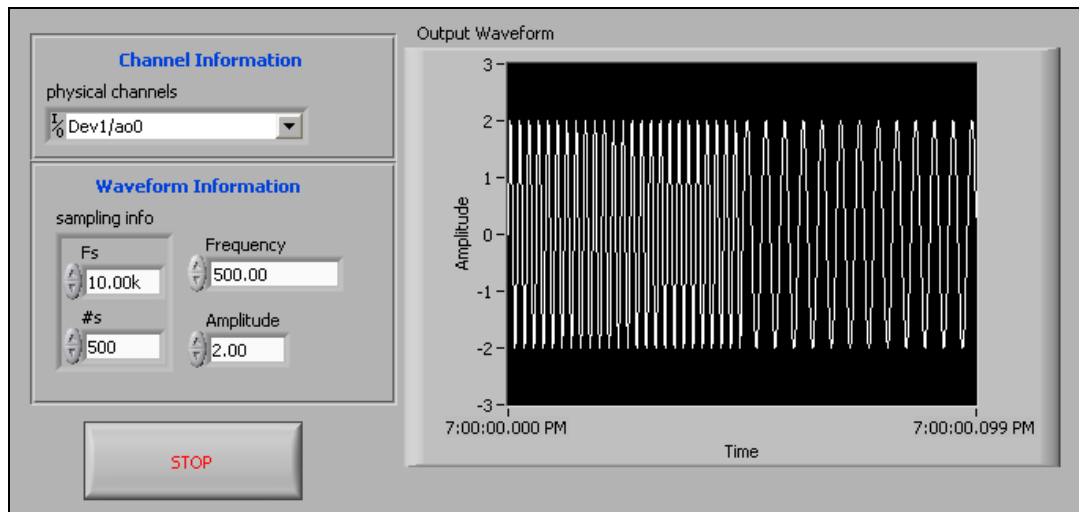
Note Substitute this exercise for Exercise 7-5 on systems that do not have amplified speakers.

To create a siren, produce two sine waves of different frequencies and continuously generate these waveforms one after the other.

1. On the DAQ Signal Accessory, connect analog out 0 to analog in 1.

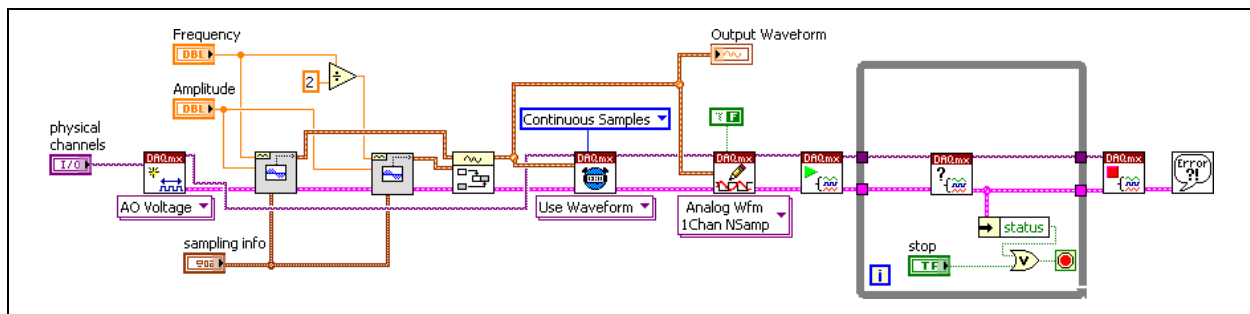
Front Panel

2. Open the Tone Generation (7-4) VI you completed in Exercise 7-4. Modify the front panel as shown in the following figure.



Block Diagram

3. Modify the block diagram as shown in the following figure.



- a. Place the DAQmx Create Virtual Channel VI, located on the **Functions»All Functions»NI Measurements»DAQmx - Data**

Acquisition palette, on the block diagram. This VI creates a virtual channel for the physical channel that you specify. Select **Analog Output»Voltage** from the pull-down menu. Right-click the **physical channels** input and select **Create»Control** from the shortcut menu.



- b. Place the Sine Waveform VI, located on the **Functions»All Functions»Analyze»Waveform Generation** palette, on the block diagram. This VI generates a sine wave. You use two of these VIs.

The leftmost Sine Waveform VI produces a sine wave with the frequency of `Frequency`. The second Sine Waveform VI produces a frequency of $1/2$ `Frequency`. The two sine waves are then appended and placed into an array. The DAQmx Write VI writes the combined waveform to the buffer.



- c. Place the Divide function, located on the **Functions»Arithmetic & Comparison»Express Numeric** palette, on the block diagram. Create a numeric constant by right-clicking the `y` input and selecting **Create»Constant** from the shortcut menu.



- d. Place the Append Waveforms VI, located on the **Functions»All Functions»Waveform»Waveform Operations** palette, on the block diagram. Appends waveform B to the end of waveform A. If the sampling rates do not match, the error cluster returns an error.



- e. Place the DAQmx Timing VI, located on the **Functions»All Functions»NI Measurements»DAQmx - Data Acquisition** palette, on the block diagram. This VI configures the task timing.



- f. Place the DAQmx Write VI, located on the **Functions»All Functions»NI Measurements»DAQmx - Data Acquisition** palette, on the block diagram. This VI writes data into the buffer for a finite, buffered analog output operation. Select the **Analog»Single Channel»Multiple Samples»Waveform** instance from the pull-down menu.



- g. Place the DAQmx Start Task VI, located on the **Functions»All Functions»NI Measurements»DAQmx - Data Acquisition** palette, on the block diagram. This VI starts the buffered analog output generation.



- h. Place the DAQmx Is Task Done VI, located on the **Functions»All Functions»NI Measurements»DAQmx - Data Acquisition»DAQmx Advanced Task Options** palette, on the block diagram. This VI queries the status of a task and returns whether it completed execution. Use this VI in the While Loop to monitor the state of the task and wait for an error to occur or for the user to click the **STOP** button.



- i. Place the DAQmx Stop Task VI, located on the **Functions»All Functions»NI Measurements»DAQmx - Data Acquisition** palette, on the block diagram. This VI stops the analog output task.
 - j. Place the Simple Error Handler VI, **located on the Functions»All Functions»Time & Dialog** palette, on the block diagram. In the event of an error, this VI displays a dialog box with information regarding the error and where it occurred.
4. Save the VI as `Siren.vi` in the `C:\Exercises\LabVIEW DAQ` directory.
 5. Open the Continuous Acquire with MIO VI located in the `C:\Exercises\LabVIEW DAQ` directory to view the waveform the Siren VI produces.
 6. Set the front panel controls of the Continuous Acquire with MIO VI as follows:
 - **Physical Channel:** `DevX/ai1`, where *X* corresponds to the device number of your DAQ device
 - **Samples per Channel:** 10000
 - **Rate:** 10000
 7. Start the Continuous Acquire with MIO VI.
 8. On the front panel of the Siren VI, set the controls as follows:
 - **Physical Channel:** `DevX/ai1`, where *X* corresponds to the device number of your DAQ device
 - **Amplitude:** 1
 - **Frequency:** 1000
 - **Sampling Info**
 - **Fs:** 10000
 - **#s:** 500

#s determines the number of samples in the waveform the Sine Waveform VI produces. This parameter is set to 1/2 the **Buffer Size** because you want a two-part waveform. If you set **#s** to 1000, each Sine Waveform VI produces 1,000 points, and you would never see the waveform the second Sine Waveform VI produces.
 9. Run the VI.

If you connected analog out 0 to a speaker, you should hear two tones alternating like a siren.
 10. Close the VIs.

End of Exercise 7-6

Exercise 7-7 Triggered Continuous Tone Generation

Objective: To build a VI to trigger analog output.

In this exercise, you modify the analog output VI from Exercise 7-5 to trigger the waveform generation using the digital trigger on the DAQ Signal Accessory.

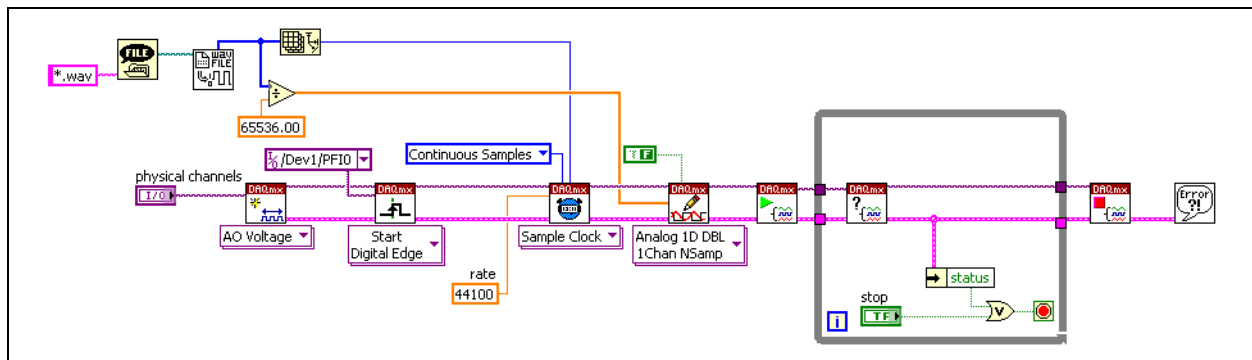


Note This exercise is designed for systems that have amplified speakers. If your system does not have amplified speakers, substitute Exercise 7-8.

1. Connect one lead from the speaker input cable to analog out 0 on the DAQ Signal Accessory. Connect the other lead from the speaker input cable to GND on the DAQ Signal Accessory. You do not need to worry about polarity.
2. Open the Continuous Tone Generation (7-5) VI you created in Exercise 7-5.
3. Select **File»Save As** and save the VI as **Triggered Continuous Tone Generation Option A.vi** in the `C:\Exercises\LabVIEW DAQ` directory.

Block Diagram

4. Modify the block diagram as shown in the following figure.



- a. Place the DAQmx Trigger VI, located on the **Functions»All Functions»NI Measurements»DAQmx - Data Acquisition** palette, on the block diagram. This VI configures the trigger conditions for the analog output task.

5. Save the VI.
6. On the front panel, set the controls as follows:

Physical channels: `DevX/ao0`, where `X` corresponds to the device number of your DAQ device.

7. Adjust the volume on your speakers to a moderate level.
8. Run the VI. The VI prompts you to open a WAV file. Select the `NISoundFile.wav` located in the `C:\Exercises\LabVIEW DAQ` directory.
9. Press the Digital Trigger button on the DAQ Signal Accessory. Remember that when you press the Digital Trigger button, it produces a falling edge. When you release the button, it produces a rising edge, which triggers the waveform generation. You should hear the WAV file through the speakers.
10. Click the **STOP** button to stop the VI.
11. Close the VI.

End of Exercise 7-7

Exercise 7-8 Digital Triggered Generation (Optional)

Objective: To create a digitally triggered analog output VI.



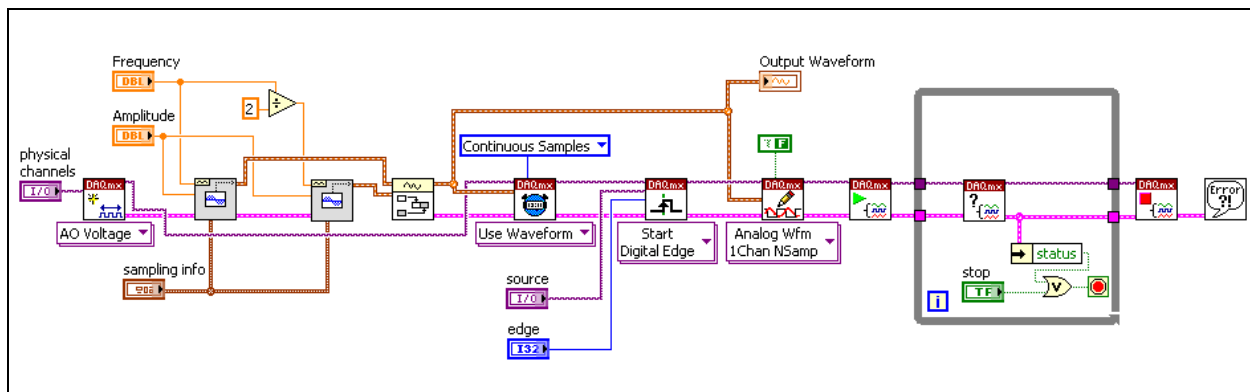
Note Substitute this exercise for Exercise 7-7 on systems that do not have amplified speakers.

In this exercise, you create a hypothetical security system that includes a siren. The same trigger for the security cameras triggers the siren. Use the Digital Trigger on the DAQ Signal Accessory to simulate this button.



Note The Digital Trigger button is wired to the PFI0 pin (E Series devices) or the EXTTRIG pin (Lab/1200 devices) on the DAQ device. Remember that the Digital Trigger button produces a falling edge when pressed and a rising edge when released.

1. On the DAQ Signal Accessory, connect analog out 0 to analog in 1.
2. Open the Siren VI you created in Exercise 7-6 and save it as `Triggered Siren.vi` in the `C:\Exercises\LabVIEW DAQ` directory.
3. Modify the block diagram as shown in the following figure.



- a. Place the DAQmx Trigger VI, located on the **Functions»All Functions»NI Measurements»DAQmx - Data Acquisition** palette, on the block diagram. This VI configures the trigger conditions for the analog output task.
4. Save the VI.
 5. Open the Continuous Acquire with MIO VI located in the `C:\Exercises\LabVIEW DAQ` directory to view the generated waveform.

6. On the front panel of the Continuous Acquire with MIO VI, set the controls as follows:
 - **Physical channels:** DevX/ai1, where X corresponds to the device number of your DAQ device
 - **Samples per Channel:** 10000
 - **Rate:** 10000
7. Run the Continuous Acquire with MIO VI.
8. On the front panel of the Triggered Siren VI, set the controls as follows:
 - **Physical Channel:** DevX/ao0, where X corresponds to the device number of your DAQ board
 - **Frequency:** 1000
 - **Amplitude:** 1
 - **Sampling Info**
 - **Fs:** 10000
 - **#s:** 500
 - **Source (trigger):** /DevX/PFI0, where X corresponds to the device number of your DAQ device
 - **Edge:** Rising
9. Run the Triggered Siren VI. Initially, the graph on the Continuous Acquire with MIO VI does not respond because the Triggered Siren VI is waiting for a trigger.
10. Press the Digital Trigger button on the DAQ Signal Accessory. Remember that when you press the Digital Trigger, it produces a falling edge. When you release the button, it produces a rising edge, which triggers the siren.

End of Exercise 7-8

Summary

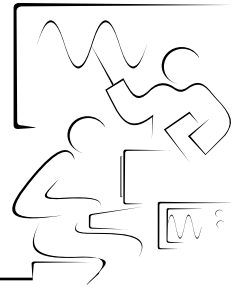
- Typical DAQ devices have a DAC for each analog output channel.
- DAQmx VIs allow single-point, buffered, continuous, or triggered generation.
- The output waveform frequency depends on the update rate and the number of cycles of the waveform in the buffer.
- To output waveform data including the waveform timing properties, use the Use Waveform instance of the DAQmx Timing VI.

Notes

Notes

Lesson 8

Digital I/O



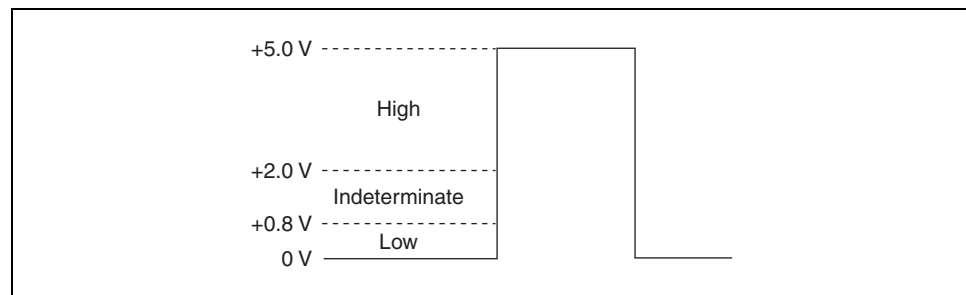
This lesson describes the digital functionality of a DAQ device, which can perform digital input and digital output.

You Will Learn:

- A. What constitutes a digital signal
- B. Digital I/O terminology
- C. Digital I/O VIs
- D. Digital Triggering

A. Digital Signals

The digital lines on a DAQ device accept and generate TTL compatible signals. A TTL signal has two states—logic low and logic high, as shown in the following illustration. Logic low signals are signals between 0 to +0.8 V. Logic high signals are signals between +2 to +5 V. Signals between +0.8 and +2.0 V are indeterminate.



To ensure that digital lines measure the signal correctly, make sure the voltage level of the signal is never between +0.8 and +2 V.

Digital Terminology

Following are some of the common terms used with digital I/O operations.

- **Bit**—The smallest unit of data used in a digital operation. Bits are binary, so they can be a 1 or a 0.
- **Line**—An individual signal in a digital port. The difference between a bit and a line is that the bit refers to the actual data transferred, and the line refers to the hardware the bit is transferred on. However, the terms line and bit are fairly interchangeable. For example, an 8-bit port is the same as a port with eight lines.
- **Port**—A collection of digital lines. Usually the lines are grouped into a 4-bit or an 8-bit port. Older DAQ devices have two 4-bit ports and most E Series devices have one 8-bit port. LabVIEW VIs often refer to a port as a digital channel.
- **Port Width**—The number of lines in a port. For example, E Series devices have one port with eight lines. Therefore, the port width is eight.
- **Mask**—Determines if a digital line is ignored. For example, if you write to a port, but you do not want to write to all the lines, you can set a mask to ignore the lines you do not want to write to.

NI-DAQmx Digital Notation

In NI-DAQmx, use the following conventions to describe digital lines and ports. In each case, *X* corresponds to the device number of your DAQ device, *Y* corresponds to a digital port, and *A* and *B* correspond to digital lines on your DAQ device.

- **Port**—Dev*X*/Port *Y*
- **Single Line**—Dev*X*/Port *Y*/Line *A*
- **Multiple Lines**—Dev*X*/Port *Y*/Line *A*:*B*. The lines are read (or written) in ascending order, starting at Line *A* and continuing to Line *B*. To explicitly control the order in which the digital lines are read (or written), use the notation Dev*X*/Port *Y*/Line *A*, Dev*X*/Port *Y*/Line *B*. The comma separates each digital line.

B. Digital I/O

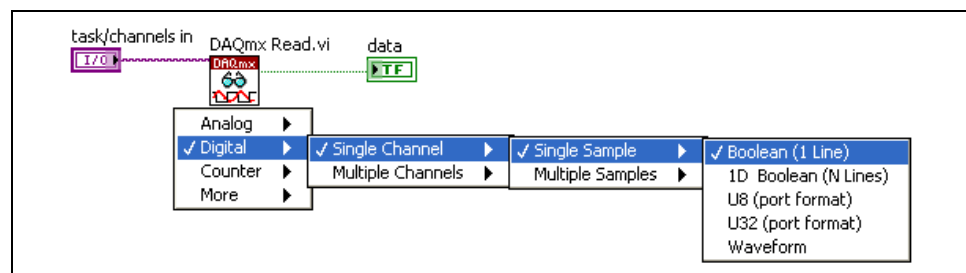
To perform digital I/O in NI-DAQmx, select a digital instance of the DAQmx Read VI or DAQmx Write VI. In addition to these VIs, use the DAQmx Timing VI and DAQmx Triggering VI to configure digital I/O tasks. To programmatically create a digital channel, use the DAQmx Create Virtual Channel VI.

DAQmx Create Virtual Channel VI

To programmatically create a digital input or output channel, select the Digital Input or Digital Output instance of the DAQmx Create Virtual Channel VI. These instances of the VI allow you to create a channel composed of a digital port, a digital line, or a collection of digital lines. In addition, use the **line grouping** input to determine the manner in which the channel organizes the digital lines. You can select to create one channel for each line or to create one channel for all lines.

DAQmx Read VI

The DAQmx Read VI reads samples from the task or channels you specify. The instances of this polymorphic VI specify what format of samples to return, whether to read a single sample or multiple samples at once, and whether to read from one or multiple channels. Select a digital instance from the pull-down menu to perform digital input.



Select to read either a single channel or multiple channels. If the channel line grouping is set to one channel for all lines, reading a single channel returns all the values on each of the lines in the channel. If the channel line grouping is set to one channel for each line, you must read multiple channels in order to read the values on each of the digital lines specified.

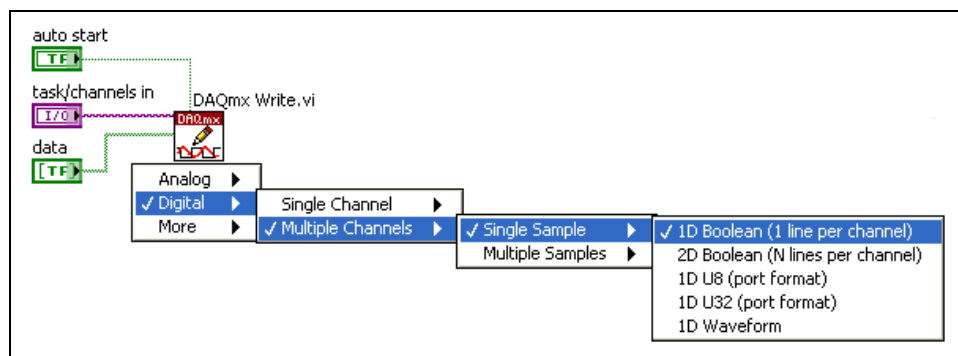
Select to read either a single sample or multiple samples at once. The data type options for the return value(s) allow you to return the value in line or port format. Line format consists of a single Boolean value or an array of Boolean values (for multiple lines). A unsigned 8-bit integer (U8) or unsigned 32-bit integer (U32) are the options for returning the value in port format. When reading multiple channels, the data type options are the same,

with an array dimension added to each type to account for the multiple channel reading.

If you select the U8 or U32 port format to return data, use the Number to Boolean Array function located on the **Numeric>Conversion** palette, to convert the number into an array of Boolean values.

DAQmx Write VI

The DAQmx Write VI writes samples to the task or channels you specify. The instances of this polymorphic VI specify the format of the samples to write, whether to write one or multiple samples, and whether to write to one or multiple channels. Select a digital instance of the DAQmx Write VI to perform digital output.



The settings for the digital instance are configured in the same manner as the digital instance of the DAQmx Read VI.

By default, the **auto start** input of the DAQmx Write VI is TRUE when writing single samples and FALSE when writing multiple samples. If you use the DAQmx Start VI and DAQmx Stop Task VI, always set the **auto start** input to FALSE. This allows you greater control of the task state model and improves the speed of your program.

DAQmx Timing VI

The handshaking instance of the DAQmx Timing VI determines the number of digital samples to acquire or generate using digital handshaking between the device and a peripheral device. Instead of specifying a sampling rate, specify the number of digital samples to acquire or generate using digital handshaking.



Note Not all devices support digital handshaking. Consult your device documentation to see if handshaking is supported on your device. For E Series devices, only those devices with more than eight digital lines—those that have an additional 8255 chip onboard—support handshaking.

DAQmx Trigger VI

Use the DAQmx Trigger VI to configure triggering for the task. The instances of this polymorphic VI correspond to the trigger and trigger type to configure. Configure digital triggers settings the same way you configure analog input and analog output triggers. Refer to Lesson 4, *Analog Input*, of this manual for more information about configuring triggers.

Exercise 8-1 Digital Writer VI

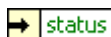
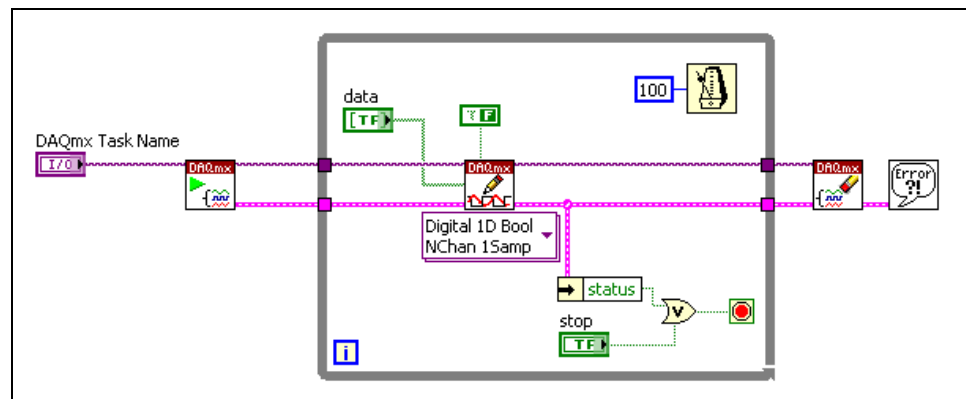
Objective: To create a digital virtual channel in LabVIEW using the DAQ Assistant and build a VI that writes to the digital lines on the DAQ Signal Accessory.

Front Panel

1. Open a blank VI and place a DAQmx Task Control on the front panel.
2. Right-click the DAQmx Task control and select **New Task (DAQ Assistant)** from the shortcut menu.
3. Configure the task with the following settings:
 - **Measurement Type:** Digital I/O
 - **Digital I/O Type:** Line Output
 - **Physical Channels:** Press and hold down the <Shift> key and select port0/line0 to port0/line3
 - **Name:** Digital Output
4. In the DAQ Assistant, click the **Test** button. The four lines on the DAQ Accessory should all appear illuminated. The lines are set to a logic low state—a zero binary value corresponds to +5 V on the digital line. Enter different binary values by clicking the radio buttons beneath the digital lines.
5. Click the **OK** button. On the **Settings** tab, click **Invert Line**.
6. Click the **Test** button again. Write different values to the digital lines and notice that the lines now have logic high settings—a zero binary value corresponds to 0 V on the digital line.
7. Click the **OK** button to exit the test panel. You may leave the digital lines inverted if you choose, depending on if you prefer a logic high or logic low state.

Block Diagram

8. Build the block diagram as shown in the following figure.



- Place the DAQmx Start Task VI, located on the **Functions»All Functions»NI Measurements»DAQmx - Data Acquisition** palette, on the block diagram. This VI starts a task.
- Place the DAQmx Write VI, located on the **Functions»All Functions»NI Measurements»DAQmx - Data Acquisition** palette, on the block diagram. Select the **Digital»Multiple Channels»Single Sample»1D Boolean** instance from the pull-down menu. This instance writes a single value to each of the digital lines included in the task.
Right-click the **data** input and select **Create»Control** from the shortcut menu.
- Place the DAQmx Clear Task VI, located on the **Functions»All Functions»NI Measurements»DAQmx - Data Acquisition** palette, on the block diagram. This VI clears all resources assigned to the task.
- Place the Simple Error Handler VI, located on the **Functions»All Functions»Time & Dialog** palette, on the block diagram. In the event of an error, this VI displays a dialog box with information about the error and where it occurred.
- Place the Unbundle By Name function, located on the **Functions»All Functions»Cluster** palette, on the block diagram.
- Place the Or function, located on the **Functions»Arithmetic & Comparison»Express Boolean** palette, on the block diagram.

9. Save the VI as `Digital Writer.vi` in the `C:\Exercises\LabVIEW DAQ` directory.
10. Switch to the front panel. Before you run the VI, you must initialize the data array. Resize the array to display four Boolean controls. The dimmed appearance of the control indicates that it is not initialized. To initialize the control, click the control and set it to either a high or low state.
11. Run the VI. Change the Boolean values in the data array and observe the digital lines on the DAQ Signal Accessory.
12. Click the **STOP** button to stop the VI.
13. Modify this VI to write values to a single digital port. Right-click the DAQmx Task Name Control and select **New Task (DAQ Assistant)** from the shortcut menu.
14. Configure the task with the following settings:
 - **Measurement Type:** Digital I/O
 - **Digital I/O Type:** Port Output
 - **Physical Channels:** Select `port0`.
 - **Name:** Digital Port Output
15. Return to the block diagram and select the **Digital»Single Channel»Single Sample»U32** instance from the DAQmx Write VI pull-down menu.
16. Place the Boolean Array to Number function, located on the **Functions»All Functions»Numeric»Conversion** palette, on the block diagram. This function converts a Boolean array to a 32-bit unsigned integer using two's complement notation. The first element of the array is the least significant bit (LSB).
17. Wire the Boolean array to the **Boolean array** input of the Boolean Array to Number function. Wire the output of this function to the **data** input on the DAQmx Write VI. Right-click the **number** output of the Boolean Array to Number function and select **Create»Indicator** from the shortcut menu.
18. Run the VI. Notice the value of the 32-bit unsigned integer in relation to the values in the Boolean array. To invert the lines on the DAQ Signal Accessory, right-click the DAQmx Task Name control, select **Edit Task (DAQ Assistant)** from the shortcut menu, and select **Invert All Lines In Port**.
19. Save and close the VI.



End of Exercise 8-1

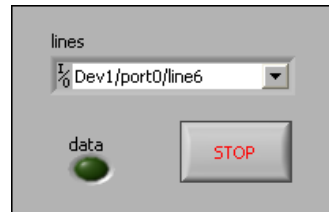
Exercise 8-2 Digital Reader VI

Objective: To build a VI that reads from a digital line on the DAQ Signal Accessory.

In this exercise, you read a value from a digital line and port using the DAQmx Read VI.

Front Panel

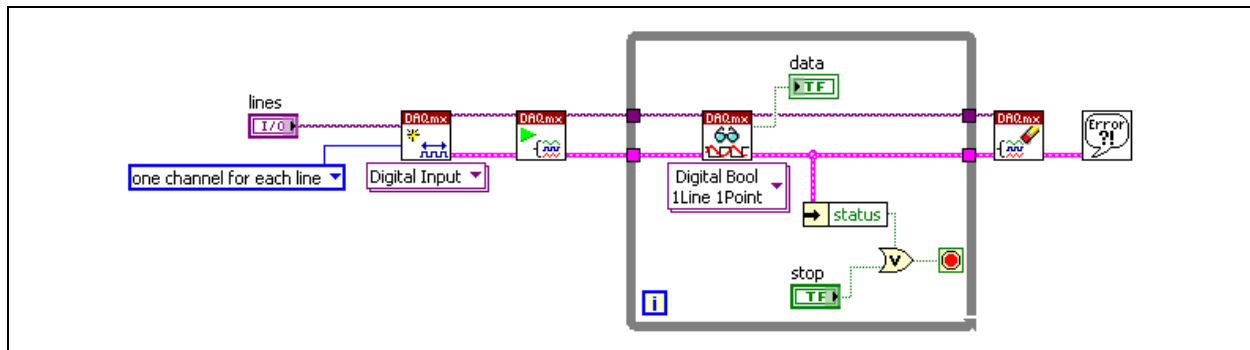
1. Open a blank VI and build the following front panel



All of the controls, including the DAQmx Physical Channel Control can be created from the block diagram by right-clicking the appropriate terminal and selecting **Create»Control** from the shortcut menu.

Block Diagram

2. Build the following block diagram.



- a. Place the DAQmx Create Virtual Channel VI, located on the **Functions»All Functions»NI Measurements»DAQmx - Data Acquisition** palette, on the block diagram. This VI creates a new digital input virtual channel. Select the **Digital Input** instance from the pull-down menu. Right-click the **line grouping** input and select **Create»Constant** from the shortcut menu.



- b. Place the DAQmx Start Task VI, located on the **Functions»All Functions»NI Measurements»DAQmx - Data Acquisition** palette, on the block diagram. This VI starts a task.



- c. Place the DAQmx Read VI, located on the **Functions»All Functions»NI Measurements»DAQmx - Data Acquisition** palette, on the block diagram. Select the **Digital»Single Channel»Single Sample»Boolean** instance from the pull-down menu. Right-click the **data** output and select **Create»Indicator** from the shortcut menu.



- d. Place the DAQmx Clear Task VI, located on the **Functions»All Functions»NI Measurements»DAQmx - Data Acquisition** palette, on the block diagram. This VI clears all resources assigned to the task.



- e. Place the Simple Error Handler VI, located on the **Functions»All Functions»Time & Dialog** palette, on the block diagram. In the event of an error, this VI displays a dialog box with information about the error and where it occurred.



- f. Place the Unbundle By Name function, located on the **Functions»All Functions»Cluster** palette, on the block diagram.



- g. Place the Or function, located on the **Functions»Arithmetic & Comparison»Express Boolean** palette, on the block diagram.

3. Save the VI as `Digital Reader.vi` in the `C:\Exercises\LabVIEW DAQ` directory.
4. On the front panel, select the line as `DevX/port0/line6`, where `X` corresponds to the device number of your DAQ device.
5. Run the VI.
6. Rotate the quadrature encoder knob on the DAQ Signal Accessory. Port 0, line 6 is hardwired to phase B of the quadrature encoder. The data LED briefly flashes when you rotate the knob.
7. Modify the VI to read the entire contents of port 0.



Tip Change to physical lines to read lines 0 through 7 and modify the instance of the DAQmx Read VI.

8. Run the VI and rotate the quadrature encoder. Which element of the Boolean array should change?
9. Save and close the VI.

End of Exercise 8-2

Summary

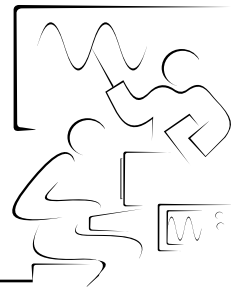
- Digital lines on a typical E Series device can:
 - Read and write TTL-compatible signals.
 - Perform only non-timed digital I/O.
 - Be configured individually for input or output.
- A typical E Series DAQ device has digital lines that can read and write TTL-compatible signals.
- The specifications for a TTL signal define 0 to +0.8 V as logic low and +2 to +5 V as logic high. Between +0.8 and +2 V can be interpreted as high or low.
- If you need to perform handshaking or any other form of timed digital I/O, you need to purchase a specialty DAQ device such as the DIO-6533.
- Use the DAQmx Read VI and DAQmx Write VI to easily configure digital input and output operations.

Notes

Notes

Lesson 9

Counters



This lesson focuses on the counter functionality of a DAQ device. It begins with an overview of counters including counter signals, the parts of a counter, the pins you connect a counter signal to, basic counter terminology, and different types of counter chips. The lesson also describes the DAQmx VIs used for counter operations.

You Will Learn:

- A. Overview of counters
- B. Edge counting
- C. Pulse generation
- D. Pulse measurement
- E. Frequency measurement
- F. Position measurement

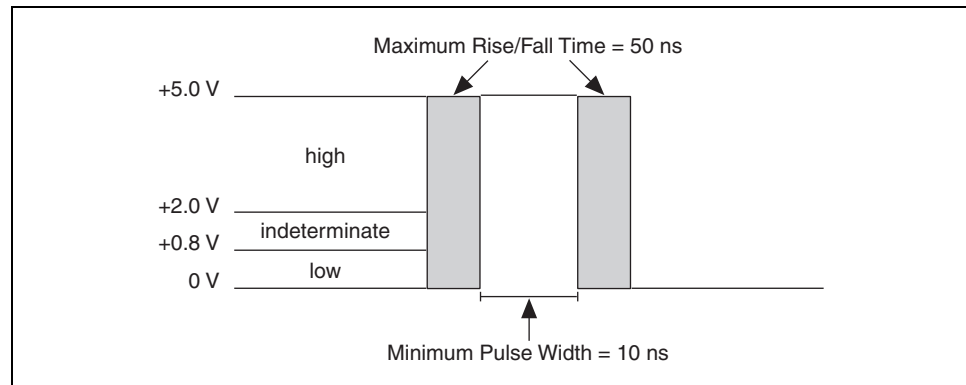
A. Counter Signals

Counters operate with TTL-compatible signals. A TTL-compatible signal has the following specifications:

$$0 \text{ V} - 0.8 \text{ V} = \text{logic low}$$

$$2 \text{ V} - 5 \text{ V} = \text{logic high}$$

$$\text{Maximum Rise/Fall Time} = 50 \text{ ns}$$



Digital I/O devices can set or monitor the state of a digital line. Counters, however, are not only concerned with the state of the signal but also with the transition from one state to another. A counter can detect rising edges (transition from logic low to logic high) and falling edges (transition from logic high to logic low). Two important parameters related to sensing rising and falling edges are the rise/fall time and the minimum pulse width. The rise/fall time is a measure of how quickly the signal transitions from low to high or high to low. For a counter to detect the edge, the transition must occur within 50 ns or less, as defined by the specifications for a TTL-compatible signal.

In addition to this time restraint, there must be a minimum delay from the time a counter detects a rising or falling edge until it can detect another rising or falling edge. This delay is known as the minimum pulse width. The minimum pulse width depends on the counter chip that is used. E Series devices have a DAQ-STC chip, which has minimum pulse width of 10 ns for both the source and the gate. Consult the hardware documentation for the specific DAQ device to determine the minimum pulse width required for the counters.

There are five different types of counter measurements—edge counting, pulse generation, pulse measurement, frequency measurement, and position measurement.

Parts of a Counter

A counter has the following main components:

- **Count Register**—Stores the current count of the counter. You can query the count register with software.
- **Source**—An input signal that can change the current count stored in the count register. The counter looks for rising or falling edges on the source signal. Whether a rising or falling edge changes the count is software-selectable. The type of edge that is selected is called the active edge of the signal. When an active edge is received on the source signal the count changes. Whether an active edge increments or decrements the current count is also software-selectable. The source signal must be TTL-compatible.
- **Gate**—An input signal that determines if an active edge on the source changes the count. Counting can occur when the gate is high, low, or between various combinations of rising and falling edges. Gate settings are made in software. The gate is similar to a line mask in digital I/O because it allows you to acknowledge or ignore active edges on the source.
- **Out**—An output signal that generates pulses or a series of pulses, otherwise known as a pulse train. The output signal is TTL-compatible.

Counter Pins

Analog input, analog output, and digital I/O all have dedicated pins for input or output operations. Counters use a combination of Programmable Function Input (PFI) pins and dedicated pins for their operations. The output pins for counters are used solely for generating pulses on the out of a counter. The source and gate pins for counters are PFI pins and can be used for applications other than the source or gate of a counter. For example, pin 3 on the 68-pin connector can be used as PFI9, the gate of counter 0, or both. The ability to use one pin for multiple applications offers a great deal of flexibility. For example, you could wire an external TTL signal into pin 3 and use it to trigger an analog input operation and be a gate for a counter operation.

Counter Terminology

The following terms are important to understand when using counters.

- **Terminal Count**—The last count before a counter reaches 0. For example, when a counter that increments the count reaches its maximum count, it has reached the terminal count. The next increment of the count forces the counter to roll over and start counting at 0.
- **Resolution**—How high the counter can count before reaching the terminal count, specified in bits. The following formula calculates the maximum count based on the resolution:

$$\text{max count} = 2^{(\text{resolution})} - 1$$

Common counter resolutions are 16-, 24-, or 32-bits.

- **Timebase**—A signal of known frequency that is provided by the DAQ device. Typical frequencies for timebases range from 100 Hz to 80 MHz. The timebase can be routed internally to the source of a counter to provide a signal of known frequency.

B. Counter Chips

Depending on the DAQ device, you could be using the DAQ-STC or NI-TIO counter chip.

DAQ-STC

The DAQ-STC is a 24-bit counter developed by National Instruments with a wide range of functionality that is used on E Series devices. The DAQ-STC can increment or decrement the count, change the count direction on-the-fly by using a hardware signal, and offers 100 kHz and 20 MHz timebases.

The DAQ-STC is more widely available on current NI devices than any of the other chips. Throughout the remainder of this lesson, the term counter refers to the DAQ-STC chip.

NI-TIO

The NI-TIO is the premier counter chip offered on NI devices. It is a 32-bit counter that is software-compatible with the DAQ-STC. The NI-TIO can increment and decrement the count, supports encoders and a hardware trigger signal, has digital filters to remove glitches; can change the frequency of a pulse train on-the-fly, and offers 100 kHz, 20 MHz, and 80 MHz timebases. The NI-TIO is used on the 660x family of devices.



Note The remainder of this lesson focuses on the use of the DAQ-STC.

C. Counter I/O

Like analog input, analog output, and digital I/O, counter operations use the DAQmx Read VI. For counter operations, select a counter instance of the DAQmx Read VI. The DAQmx Write VI is not used with counters, as you will see in later exercises in this lesson. The DAQmx Create Virtual Channel VI, DAQmx Timing VI and DAQmx Triggering VI are also used to configure counter measurements or generations.

DAQmx Create Virtual Channel VI

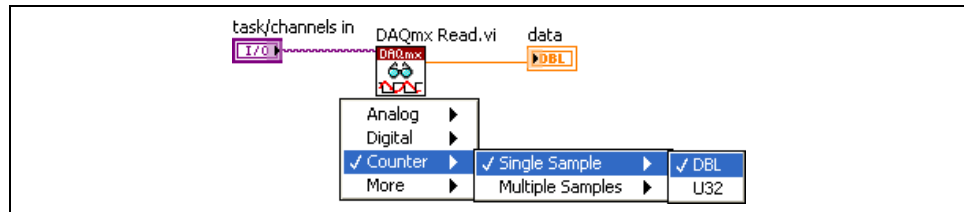
To programmatically create a counter input or output channel, select the Counter Input or Counter Output instance of the DAQmx Create Virtual Channel VI.

A Counter Input channel allows you to measure either frequency, period, count edges, pulse width, or semi period.

The configuration options for a Counter Output channel allow you to generate a pulse in terms of frequency, time, or ticks.

DAQmx Read VI

To read a sample or samples from a counter task, select a counter instance from the DAQmx Read VI pull-down menu. For counters, you can read only a single channel at a time, so the single or multiple channel selection window is no longer available.



Select to read either a single sample or multiple samples at once. When reading single samples, select to return the data as either a double-precision, floating-point numeric or unsigned 32-bit integer (U32). Multiple samples return as a 1D array of double-precision, floating-point numerics or a 1D array of unsigned 32-bit integers.

DAQmx Timing VI

For counter operations, select the Sample Clock or Implicit instances of the DAQmx Timing VI. The Sample Clock instance allows you to configure the actual timing rates. The Implicit instance sets only the number of samples to acquire or generate without specifying timing. You will use the Implicit instance of the DAQmx Timing VI later in this lesson when generating pulse trains.

DAQmx Trigger VI

Use the DAQmx Trigger VI to configure triggering for the task. The instances of this polymorphic VI correspond to the trigger and trigger type to configure. Configure counter triggers settings the same way you configure analog input and analog output triggers. In addition, use the DAQmx Trigger Property Node to configure settings for a pause trigger. Refer to Lesson 4, *Analog Input*, of this manual for more information about configuring triggers.

D. Edge Counting

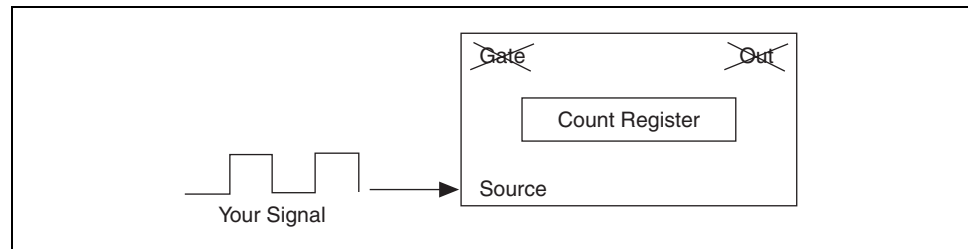
Edge counting is the most basic counter operation. In edge counting, the focus is on measuring the source signal. This section describes simple edge counting and time measurement.

Simple Edge Counting

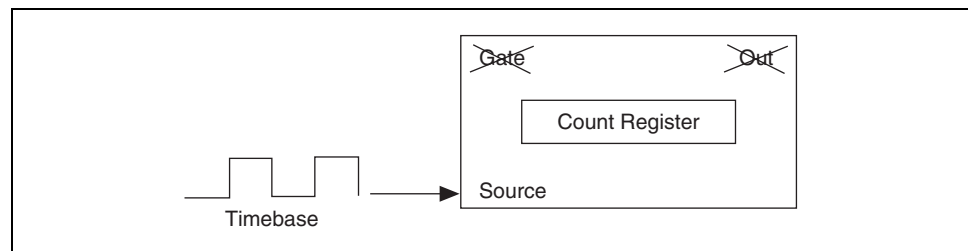
Simple edge counting fits into the basic definition of a counter. The active edges of the source signal increment the value of the count register.

An active edge can be software-selected to be a falling or rising edge.

The gate and out are not used for simple edge counting.



Time measurement is a variation on simple edge counting. When you perform simple edge counting, the source is the unknown. You use the counter to help you measure the source. When you perform time measurement, the source is a timebase of known frequency. You can use your knowledge of the timebase frequency to help you measure elapsed time. Time measurement uses a timebase for the source instead of using the signal you are trying to measure.



The following formula calculates the elapsed time:

$$\text{elapsed time} = (\text{count register value}) \times (\text{timebase period})$$

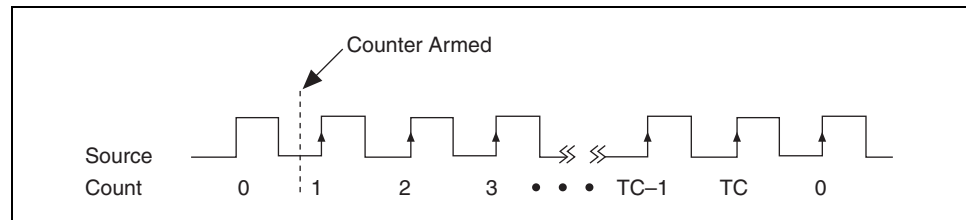
where timebase period = 1/timebase frequency.

The only difference between time measurement and simple edge counting is the signal that is used for the source.

Time Measurement

When a counter is configured for simple edge counting or time measurement, the count increments when an active edge is received on the source. You can use LabVIEW to specify if the active edge is rising or falling.

In the following example, the rising edge was selected as the active edge. The count increments by one every time a rising edge is reached.



Notice that the count cannot increment until the counter has been armed (started). A counter has a fixed number it can count to as determined by the resolution of the counter. For example, a 24-bit counter can count to:

$$2(\text{Counter resolution}) - 1 = 2^{(24)} - 1 = 16,777,215$$

When a 24-bit counter reaches the value of 16,777,215, it has reached the terminal count. The next active edge will force the counter to roll over and start at 0.

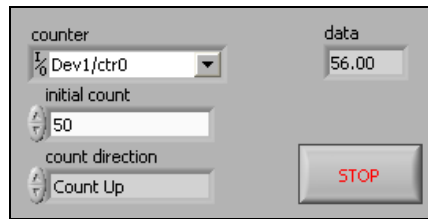
Exercise 9-1 Simple Edge Counting

Objective: To build a VI that performs simple edge counting.

In this exercise, you will build a VI that records the number of times an edge occurs.

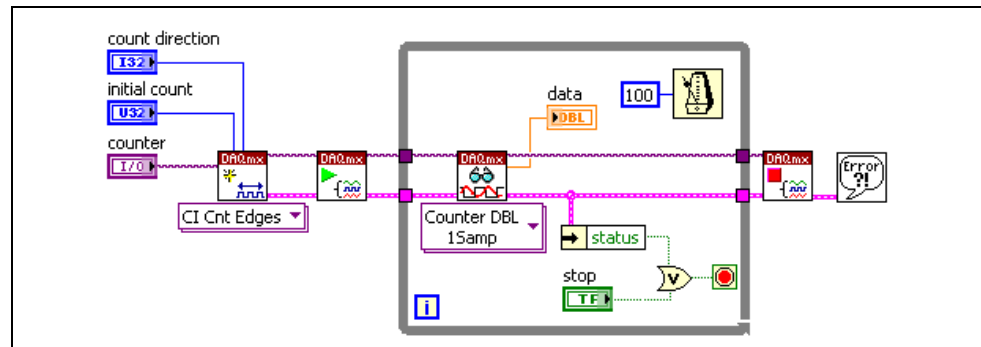
Front Panel

1. Open a blank VI. The following front panel results from building the block diagram.



Block Diagram

2. Build the following block diagram.



- a. Place the DAQmx Create Virtual Channel VI, located on the **Functions»All Functions»NI Measurements»DAQmx - Data Acquisition** palette, on the block diagram. This VI creates a new virtual channel. Select the **Counter Input»Count Edges** instance from the pull-down menu.

Right-click the **count direction**, **initial count**, and **counter inputs** and select **Create»Control** from the shortcut menu.



- b. Place the DAQmx Start Task VI, located on the **Functions»All Functions»NI Measurements»DAQmx - Data Acquisition** palette, on the block diagram. This VI starts a task.



- c. Place the DAQmx Read VI, located on the **Functions»All Functions»NI Measurements»DAQmx - Data Acquisition** palette, on the block diagram. Select the **Counter»Single Sample»DBL** instance from the pull-down menu to read a single double-precision, floating-point value from the counter.

Right-click the **data** output and select **Create»Indicator** from the shortcut menu.



- d. Place the DAQmx Stop Task VI, located on the **Functions»All Functions»NI Measurements»DAQmx - Data Acquisition** palette, on the block diagram. This VI stops the task.



- e. Place the Wait Until Next ms Multiple function, located on the **Functions»All Functions»Time & Dialog** palette, on the block diagram.



- f. Place the Unbundle by Name function, located on the **Functions»All Functions»Cluster** palette, on the block diagram.



- g. Place the Or function, located on the **Functions»Arithmetic & Comparison»Express Boolean** palette, on the block diagram.



- h. Place the Simple Error Handler VI, located on the **Functions»All Functions»Time & Dialog** palette, on the block diagram.

3. Save the VI as `Simple Edge Counting.vi` in the `C:\Exercises\LabVIEW DAQ` directory.
4. Wire channel A from the quadrature encoder on the DAQ Signal Accessory to Counter 0 Source for E-Series devices.
5. On the front panel, set the controls with the following values:
 - **Counter:** `DevX/ctr0`, where *X* corresponds to the device number of your DAQ device
 - **Initial Count:** 0
 - **Count Direction:** `Count Up`
6. Run the VI. Rotate the quadrature encoder on the DAQ Signal Accessory. The **Count** indicator should increase in value. The quadrature encoder generates a pulse that feeds into the source of counter 0.
7. Stop and close the VI.

End of Exercise 9-1

E. Advanced Edge Counting

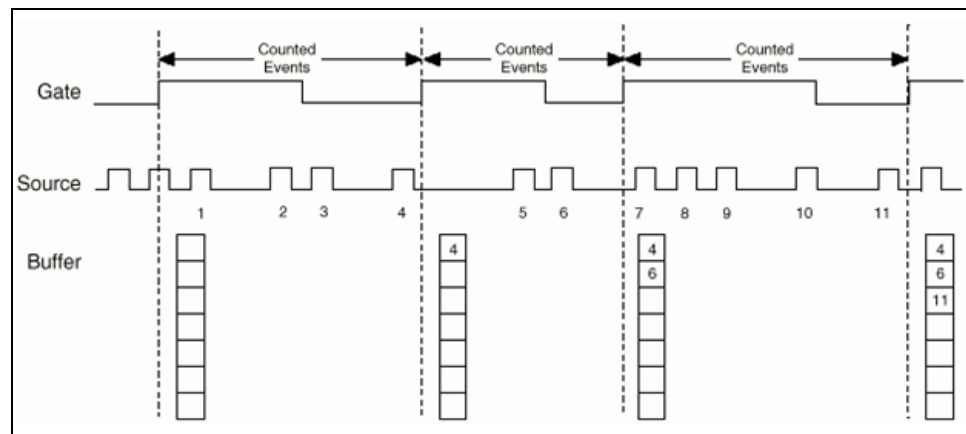
In addition to performing simple edge counting, NI-DAQmx can be easily configured to perform more advanced edge counting methods. These methods include pause trigger (gated) counting, and continuous and finite buffered counting.

Pause Trigger (Gated) Counting

In pause triggering, also known as gated triggering, an additional TTL signal enables/disables the count register. The counter value will increase when the gate level is either high or low, depending on the configuration settings you choose with the DAQmx Trigger property node.

Continuous Buffered Edge Counting

In continuous buffered edge counting, an additional TTL signal “latches” the current count register value into a buffer. Thus, the value in the buffer is only updated on the gate’s active edge. The following illustration demonstrates this transfer of the count register into the buffer.



Buffered edge counting is useful to measure the elapsed time between sequential edges occurring on the counter’s gate. Active edges on the gate latch the current counter register values into PC memory. Using either interrupts or DMA (software configurable with DAQmx Channel Property node), count register values are transferred individually to a software buffer across the PCI bus.

Finite Buffered Edge Counting

Finite buffered edge counting follows the same method for data transfer as continuous buffered edge counting, except that only a finite number of counts are acquired. As you will see in the following exercise, use the DAQmx Timing VI to set the number of samples to acquire.

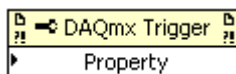
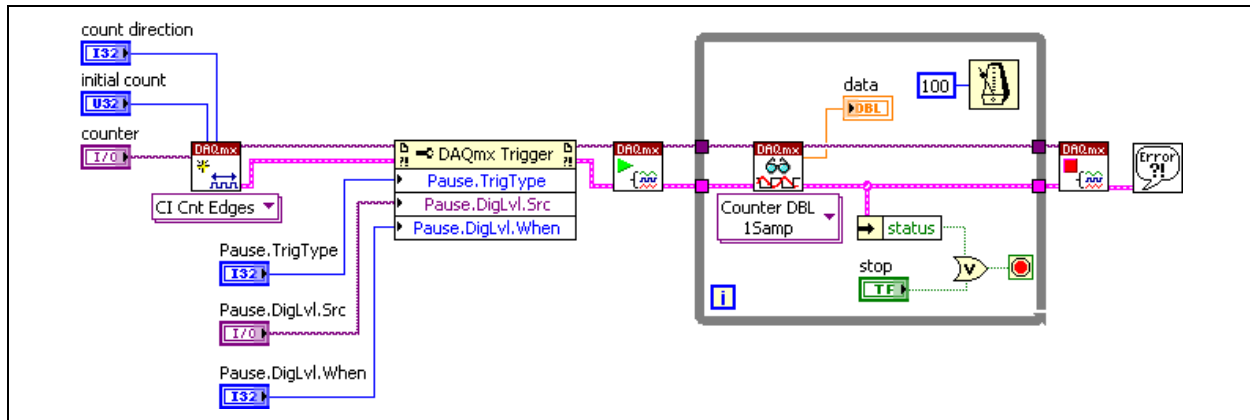
Exercise 9-2 Advanced Edge Counting

Objective: To use pause trigger, continuous buffered, and finite buffered methods to perform edge counting.

Pause Trigger (Gated) Counting

In pause trigger, or gated, counting, we use an additional line to control the counter's gate. The gate will pause the increment or decrement of the counter when the gate is either in a high or low state (depending on the settings you choose).

1. Open the Simple Edge Counting VI located in the C:\Exercises\LabVIEW DAQ directory.
2. Select **File»Save As** and save the VI as Simple Edge Counting - Gated.vi in the C:\Exercises\LabVIEW DAQ directory.
3. Modify the block diagram as shown in the following figure.



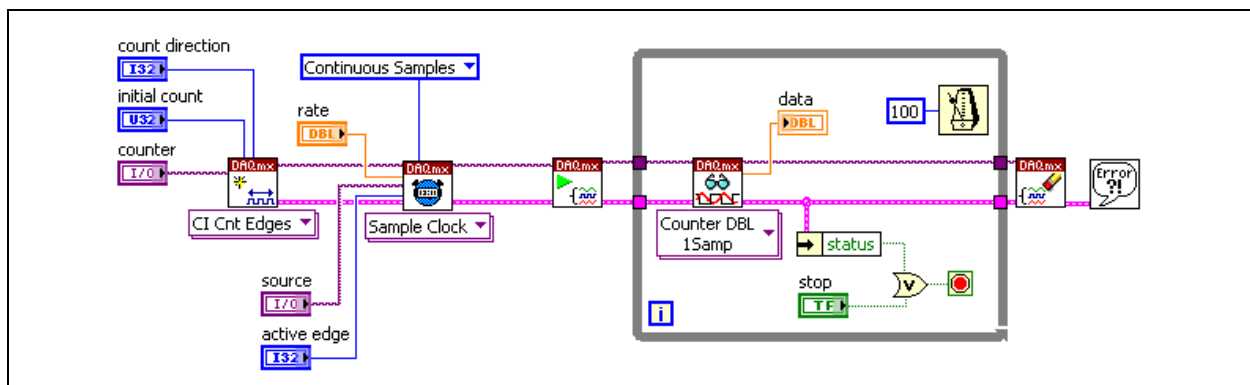
- Place the DAQmx Trigger Property Node, located on the **Functions»All Functions»NI Measurements»DAQmx - Data Acquisition** palette, on the block diagram. Use this Property Node to perform additional configuration of the task trigger. Resize the Property Node to contain three properties. Right-click each terminal and select the following properties from the shortcut menu:
 - **Properties»More»Pause»Trigger Type**
 - **Properties»More»Pause»Digital Level»Source**
 - **Properties»More»Pause»Digital Level»Pause When**
- For each property, right-click the terminal and select **Create»Control** from the shortcut menu.

4. Switch to the front panel and select the following values for the new controls.
 - **Trigger Type:** Digital
 - **Source:** /DevX/PFI0, where *X* corresponds to the device number of your DAQ device. PFI0 corresponds to the Digital Trigger button on the DAQ Signal Accessory.
 - **Pause When:** Low
5. Run the VI. Rotate the quadrature encoder knob. To pause the counting, hold down the Digital Trigger button. While you press the Digital Trigger button, rotate the quadrature encoder knob and notice that the count value remains unchanged.
6. Stop the VI and change the **Pause When** value to High.
7. Run the VI and observe the behavior with this trigger setting.
8. Save and close the VI.

Continuous Buffered Counting

In continuous buffered counting, the gate source determines when to transfer the current count value into the count register in the onboard memory. To perform this type of counting, add and configure a DAQmx Timing VI to the Simple Edge Counting VI.

1. Open the Simple Edge Counting VI located in the C:\Exercises\LabVIEW DAQ directory.
2. Select **File>Save As** and save the VI as Simple Edge Counting - Cont Buffered.vi in the C:\Exercises\LabVIEW DAQ directory.
3. Modify the block diagram to include the DAQmx Timing VI as shown in the following figure.

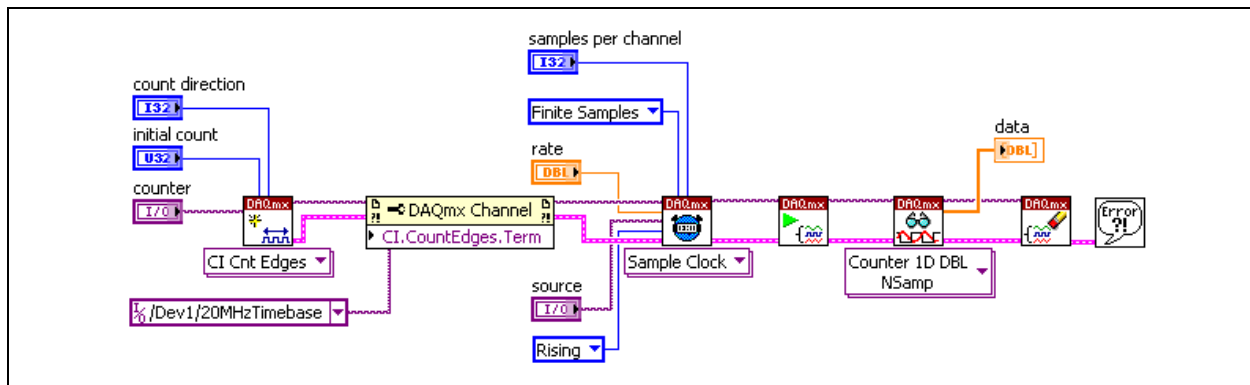


4. Switch to the front panel and select the following values for the new controls:
 - **Source:** /DevX/PFI0, where *X* corresponds to the device number of your DAQ device
 - **Active Edge:** Rising
 - **Rate:** 10000
5. Run the VI. When you press the Digital Trigger button, the current count value becomes latched into the count register. Rotate the knob on the quadrature encoder and notice that the data indicator does not change. Press the Digital Trigger button to latch and read the counter.
6. Save and close the VI.

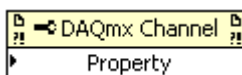
Finite Buffered Counting

In finite buffered counting, you specify the total number of edges to count. Counting stops when this number is reached.

1. Open the Simple Edge Counting VI located in the C:\Exercises\LabVIEW DAQ directory.
2. Select **File>Save As** and save the VI as Simple Edge Counting - Finite Buffered.vi in the C:\Exercises\LabVIEW DAQ directory.
3. Modify the block diagram as shown in the following figure.



- a. Right-click the While Loop and select **Remove While Loop** from the shortcut menu.
- b. Place the DAQmx Channel Property Node, located on the **Functions>All Functions>NI Measurements>DAQmx - Data Acquisition** palette, on the block diagram. Use this Property Node to configure additional settings about a channel. Right-click the terminal and select the **Properties>Counter Input>Count Edges>Input Terminal** property from the shortcut menu. Right-click the terminal, select **Create>Constant** from the shortcut menu, and



select the 20 MHz Timebase for your DAQ device. Instead of counting the clicks on the quadrature encoder, you will count the edges of the internal timebase.

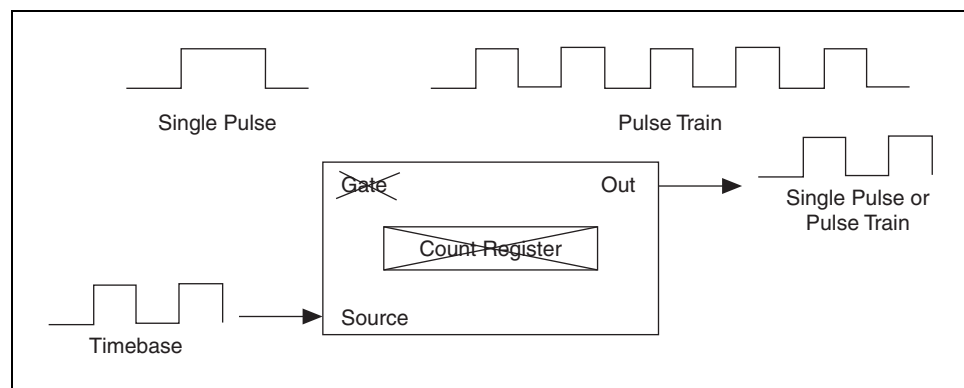


- c. Place the DAQmx Clear Task VI, located on the **Functions»All Functions»NI Measurements»DAQmx - Data Acquisition** palette, on the block diagram. This VI clears all resources assigned to the task.
4. Switch to the front panel and select the following values for the new controls:
 - **Samples per Channel:** 1000
 - **Rate:** 1000
 - **Source:** /DevX/PFI9, where *X* corresponds to the device number of your DAQ device. PFI9 is the default pin for the Counter 0 Gate.
5. Wire the square wave of the function generator to Counter 0 Gate.
6. Run the VI. You will see the latched value of the register at every rising edge of the gate signal.
7. Save and close the VI.

End of Exercise 9-2

F. Pulse Generation

A counter not only measures TTL signals, but it also generates TTL signals. Using a counter to generate a TTL signal is known as pulse generation. The output signal shown in the following illustration is generated on the out of the counter. The generated signal can be a single pulse or a continuous set of pulses known as a pulse train. The counter uses a timebase as the source to help generate the pulse. For now, you only need to understand that a timebase helps generate the pulse. Knowing how the timebase generates a pulse is not necessary to program the counter for pulse generation and is beyond the scope of this course.



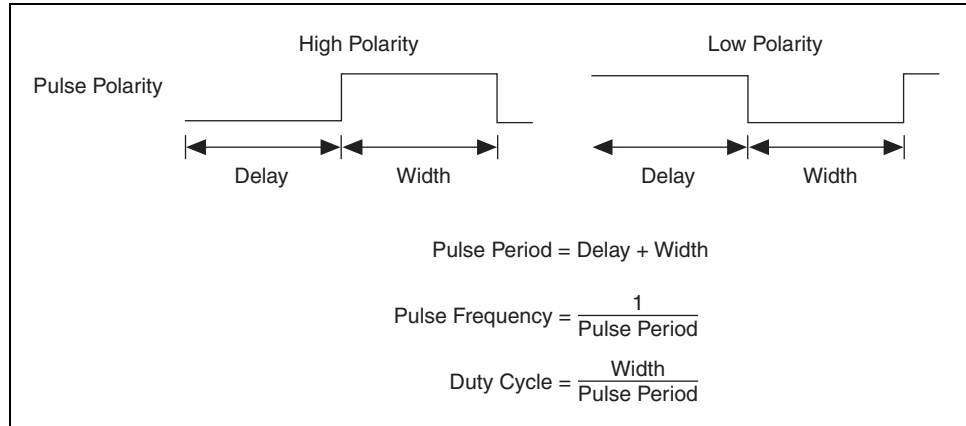
Pulse Characteristics

To generate a pulse, you must understand certain characteristics of a pulse. A pulse has two parts: the delay and the width. The delay is the first phase of the pulse, and the width is the second phase of the pulse. The delay and width are always at opposite logic levels.

For example, if the delay is logic low, the width must be logic high. A pulse can be characterized as high or low polarity. A high polarity pulse has a delay that is logic low and a width that is logic high. A low polarity pulse has a delay that is logic high and a width that is logic low. The naming convention for the pulse polarity corresponds to the logic level of its width. The period of a pulse is the time it takes the pulse to complete one cycle, so by adding the delay time to the width time, you can obtain the pulse period. After you have determined the period of the pulse, take the inverse to obtain the frequency of the pulse.

The delay and the width of a pulse are not always equal, so you need a property of a pulse that helps you determine if the delay is larger than the width or vice versa. The parameter you use is called the duty cycle. The following illustration shows the formula. The duty cycle gives you a number between 0 and 1. This number is often converted into a percentage. A pulse where the delay is equal to the width will have a duty cycle of 0.5, or 50%.

A duty cycle greater than 50% means the width is larger than the delay, and a duty cycle less than 50% means the delay is larger than the width.



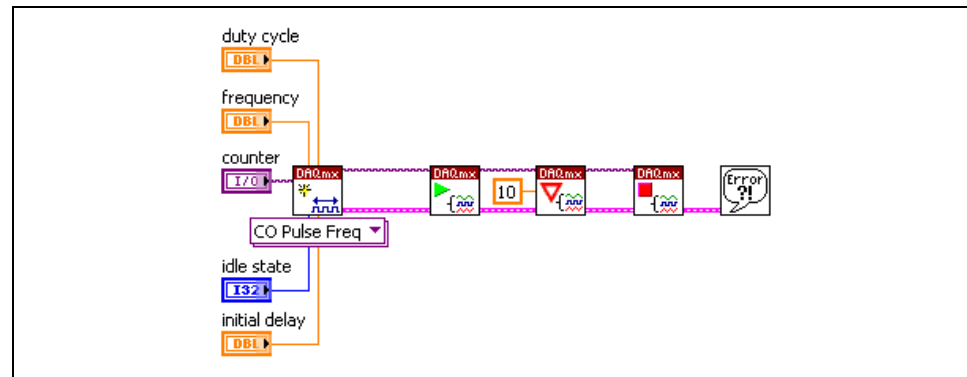
Exercise 9-3 Pulse Generation

Objective: To build a VI that generates a single pulse using a counter.

This VI demonstrates how to output a value to a counter. Although you output a frequency in this exercise, you also can output counter ticks and time with the same concept presented in this exercise.

Block Diagram

1. Open a blank VI and build the following block diagram.



- a. Place the DAQmx Create Virtual Channel VI, located on the **Functions»All Functions»NI Measurements»DAQmx - Data Acquisition** palette, on the block diagram. This VI creates a new virtual channel. Select the **Counter Output»Pulse Generation»Frequency** instance from the pull-down menu.

Right-click the **duty cycle**, **frequency**, **counter**, **idle state**, and **initial delay** inputs and select **Create»Control** from the shortcut menu.



- b. Place the DAQmx Start Task VI, located on the **Functions»All Functions»NI Measurements»DAQmx - Data Acquisition** palette, on the block diagram. This VI starts a task.



- c. Place the DAQmx Wait Until Done VI, located on the **Functions»All Functions»NI Measurements»DAQmx - Data Acquisition** palette, on the block diagram. This VI waits for the generation to complete. Use this VI to ensure that the counter output completes before the task stops.

Right-click the **timeout** input and select **Create»Constant** from the shortcut menu. The default value is 10 seconds.



- d. Place the DAQmx Stop Task VI, located on the **Functions»All Functions»NI Measurements»DAQmx - Data Acquisition** palette, on the block diagram. This VI stops the task.

2. Save the VI as `Single Pulse Generation.vi` in the `C:\Exercises\LabVIEW DAQ` directory.
3. On the front panel of the VI, modify the following controls:
 - **Counter:** `/DevX/ctr1`, where *X* corresponds to the device number of your DAQ device
 - **Duty Cycle:** 0.5
 - **Frequency:** 0.5
 - **Idle State:** Low
 - **Initial Delay:** 0.25
4. Wire counter 1 out to counter 0 source and analog in 1 on the DAQ Signal Accessory.
5. Open the Continuous Acquire with MIO VI located in the `C:\Exercises\LabVIEW DAQ` directory.
6. Set the controls on the front panel of the Continuous Acquire with MIO VI with the following values:
 - **Physical Channels:** `DevX/ai1`, where *X* corresponds to the device number of your DAQ device
 - **Samples per Channel:** 1000
 - **Rate:** 10000
7. Run the Continuous Acquire with MIO VI.
8. Run the Single Pulse Generation VI. You should see the pulse appear on the Continuous Acquire with MIO VI. Notice that the signal starts low, goes high, and returns to low.



Note It is easier to see the pulse if you disable Autoscale Y by right-clicking the graph while the VI is running and selecting **AutoScale Y** from the shortcut menu to remove the checkmark from the item. A good y-scale to see the pulse is -2 to 6.

9. Stop the Continuous Acquire with MIO VI.
10. Open the Simple Edge Counting VI located in the `C:\Exercises\LabVIEW DAQ` directory.
11. Set the **Counter** to `/DevX/ctr0`, where *X* corresponds to the device number of your DAQ device.
12. Run the Simple Edge Counting VI.
13. Run the Single Pulse Generation VI and notice that the Count increases on the Simple Edge Counting VI.
14. Close all VIs when you finish. Do not save changes.

End of Exercise 9-3

Exercise 9-4 Pulse Train Generation

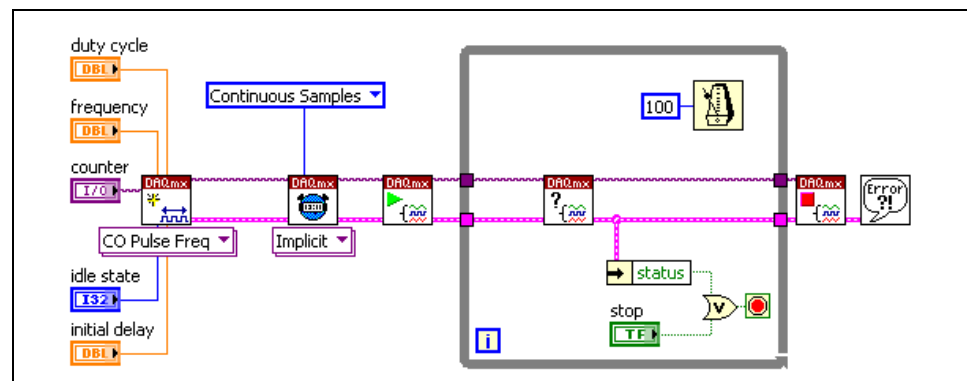
Objective: To build a VI that generates a pulse train.

In this exercise, you will modify the Single Pulse Generation VI to generate a pulse train or a series of pulses.

1. Open the VI Single Pulse Generation VI located in the C:\Exercises\LabVIEW DAQ directory.

Block Diagram

2. Modify the block diagram as shown in the following figure.



- a. Place the DAQmx Timing VI, located on the **Functions»All Functions»NI Measurements»DAQmx - Data Acquisition** palette, on the block diagram. Select the Implicit instance from the pull-down menu to configure timing so the task generates samples without specifying timing. A pulse train generation is ideal for the Implicit timing, because the pulse train itself contains all the timing parameters.

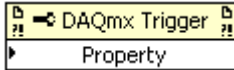


- b. Place the DAQmx Is Task Done VI, located on the **Functions»All Functions»NI Measurements»DAQmx - Data Acquisition»DAQmx Advanced Task Options** palette, on the block diagram. This VI waits for the generation to complete. Use this VI to ensure that the counter output completes before the task stops.

3. Set the front panel controls with the following values.
 - **Counter:** /DevX/ctr1, where X corresponds to the device number of your DAQ device
 - **Duty Cycle:** 0.5
 - **Frequency:** 0.5
 - **Idle State:** Low
 - **Initial Delay:** 0.25

4. Wire counter 1 out on the DAQ Signal Accessory to analog in 1.
5. Select **File»Save As** and save the VI as `Pulse Train Generator.vi` in the `C:\Exercises\LabVIEW DAQ` directory.
6. Open the Continuous Acquire with MIO VI located in the `C:\Exercises\LabVIEW DAQ` directory.
7. Set the controls on the front panel of the Continuous Acquire with MIO VI with the following values:
 - **Physical Channels:** `DevX/ai1`, where *X* corresponds to the device number of your DAQ device
 - **Samples per Channel:** 1000
 - **Rate:** 10000
8. Run the Continuous Acquire with MIO VI.
9. Run the Pulse Train Generator VI. You should see the pulse train appear on the graph of the Continuous Acquire with MIO VI.
10. Stop and close the Pulse Train Generator VI.

End of Exercise 9-4

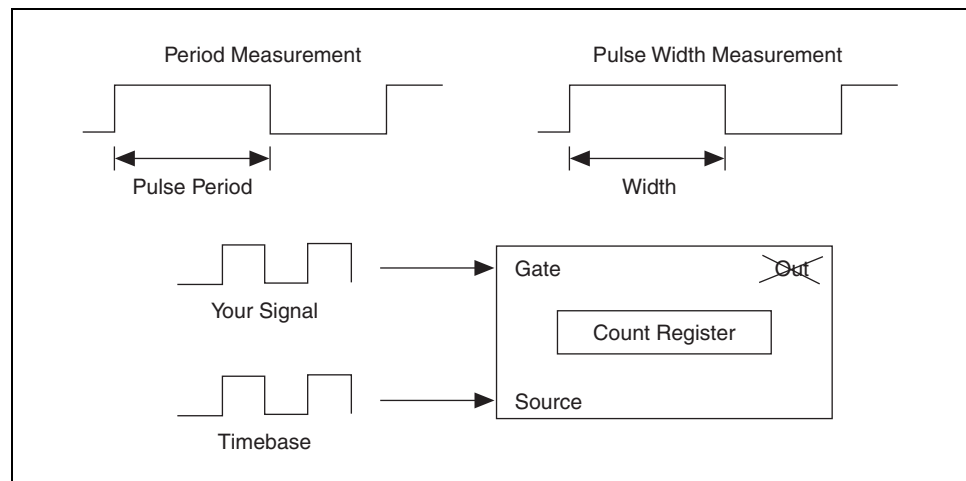


- c. Place the DAQmx Trigger Property Node, located on the **Functions»All Functions»NI Measurements»DAQmx - Data Acquisition** palette, on the block diagram. Use this Property Node to configure additional settings for the task triggering. Right-click the terminal and select **Properties»Start»More»Retriggerable** from the shortcut menu. Right-click the Property Node and select **Create»Constant** from the shortcut menu. Set this Boolean constant to True to allow the pulse train to be retriggerable.
4. Switch to the front panel and set the controls with the following values:
 - **Counter:** DevX/ctr0, where X corresponds to the device number of your DAQ device
 - **Duty Cycle:** 0.5
 - **Frequency:** 5
 - **Idle State:** Low
 - **Initial Delay:** 0
 - **Samples per channel:** 5
5. Wire counter 0 out to analog in 1 on the DAQ Signal Accessory.
6. Launch MAX and open a test panel for your NI-DAQmx device. Click the **Analog Input** tab and select the appropriate channel name. Set the **Acquisition Mode** to Continuous, the **Rate** to 50.00, and the **# Points to Read** to 50. Click the **Start** button.
7. Run the VI. Press the Digital Trigger button on the DAQ Signal Accessory to trigger the pulse generation. Watch the test panel and count the number of pulses. The number of pulses should be equivalent to the samples per channel control value.
8. Press the Digital Trigger button again. The pulse train is produced each time the trigger is enabled, making it a retriggerable pulse train.
9. Stop the VI and the test panel. Exit MAX.
10. Save and close the VI.

End of Exercise 9-5

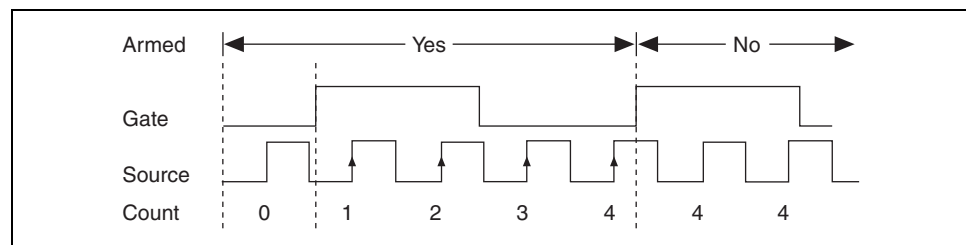
G. Pulse Measurement

When you measure a pulse, the signal you are measuring is used as the gate and the source is a timebase of known frequency, as shown in the following illustration. You can use the known frequency of the timebase and the value of the count register to determine characteristics of the gate pulse, such as pulse period and pulse width.



Period Measurement

Period measurement is one type of pulse measurement. With period measurement, you still count the active edges on the source signal. However, unlike simple edge counting, you only increment the count during the period of the gate signal. The following illustration shows period measurement that is started and stopped by a rising edge on the gate signal.



You also can start and stop counting between falling edges. The count reflects the number of rising edges on the source between the two rising edges on the gate. Therefore, to perform period measurement, you need a signal with two rising edges or two falling edges. A single pulse has only one rising edge and one falling edge, so you would not be able to measure its period.

In the previous example, one period of the gate signal has four counts. Remember that the source is a timebase of known frequency. Assume that you are using a 100 kHz timebase. The formula for calculating the period of the gate is as follows:

$$\text{pulse period} = \text{count} \times (1/\text{source frequency})$$

For the previous example, with the 100 kHz timebase, the formula yields:

$$\text{pulse period} = 4 \times (1/100,000) = 0.04 \text{ milliseconds}$$

Semi Period Measurement

Semi period measurement is very similar to period measurement, but we are only measuring the time between consecutive edges. The formula for calculating the semi period is as follows:

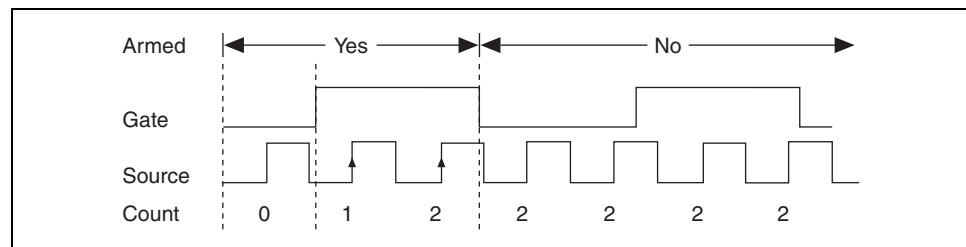
$$\text{pulse period} = \text{count} \times (1/(2 \times \text{source frequency}))$$

For the previous example, with the 100 kHz timebase, the formula yields:

$$\text{pulse period} = 4 \times (1/200,000) = 0.02 \text{ milliseconds}$$

Pulse Width Measurement

Pulse width measurement is very similar to period measurement. The difference is where you stop counting. With period measurement, you started and stopped counting with two rising edges on the gate signal. With pulse width measurement, you count only during the pulse width, so you start counting on one edge and end on the opposite edge. The count value increments only between two opposite edges, as shown in the following illustration.



The formula for calculating the pulse width is the same as the formula for the period:

$$\text{pulse width} = \text{count} \times (1/\text{source frequency})$$

For the previous example, a 100 kHz source yields:

$$\text{pulse width} = 2 \times (1/100,000) = 0.02 \text{ milliseconds}$$

0.02 milliseconds is half the value that you received for period measurement, so you must have a gate signal with a duty cycle of 50%.

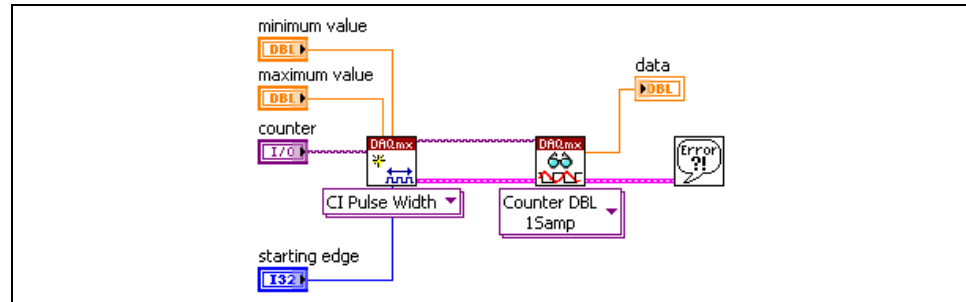
Exercise 9-6 Period, Semi Period and Pulse Width Measurement

Objective: To build a VI to measure a pulse, period, and semi period.

1. Open a blank VI and switch to the block diagram.

Pulse Width Measurement

2. Create the following block diagram.



3. Save the VI as `Signal Measurements.vi` in the `C:\Exercises\LabVIEW DAQ` directory.
4. Wire Counter 0 Out to Counter 1 Gate on the DAQ Signal Accessory.
5. Open the Single Pulse Generation VI located in the `C:\Exercises\LabVIEW DAQ` directory.
6. Set the controls on the front panel of the Single Pulse Generation VI with the following values:
 - **Counter:** `DevX/ctr0`, where `X` corresponds to the device number of your DAQ device
 - **Duty Cycle:** 0.5
 - **Frequency:** 1
 - **Idle State:** Low
 - **Delay:** 0.5
7. Set the controls on the front panel of the Signal Measurements VI with the following values:
 - **Counter:** `DevX/ctr1`, where `X` corresponds to the device number of your DAQ device
 - **Starting edge:** Rising
 - **Maximum value:** 0.001000
 - **Minimum value:** 0.000001



Note The signal you are measuring must start low when you are measuring a high pulse. Likewise, the signal must start high when you are measuring a low pulse. If the signal starts high when you are measuring a high pulse, or vice versa, an error results.

8. Run the Signal Measurements VI.
9. Run the Single Pulse Generation VI.

You should see that the Signal Measurements VI measured the 0.5 second pulse the Single Pulse Generation VI generates.

Period Measurement

1. Close the Single Pulse Generation VI and open the Pulse Train Generator VI that you completed in Exercise 9-4.
2. Modify the Signal Measurements VI to read the Period instead of the Pulse Width. Create a control for the **Measurement Time** and set it to 0.05.
3. Set the controls on the front panel of the Pulse Train Generator VI with the following values:
 - **Counter:** /DevX/ctr0, where X corresponds to the device number of your DAQ device
 - **Duty Cycle:** 0.5
 - **Frequency:** 2
 - **Idle State:** Low
 - **Initial Delay:** 0.25
4. Run the Pulse Train Generator VI.
5. Run the Signal Measurements VI. Notice that you measure a period of 0.5 seconds.
6. Stop the Pulse Train Generator VI.

Semi Period Measurement

1. Modify the Signal Measurements VI to read the semi period instead of the Period.
2. Run the Pulse Train Generator VI, leaving the control values the same as for the period measurement.
3. Run the Signal Measurements VI. The semi period is 0.25 seconds.
4. Stop the Pulse Train Generator VI.
5. Save the Signal Measurements VI.
6. Close all VIs.

End of Exercise 9-6

H. Frequency Measurements

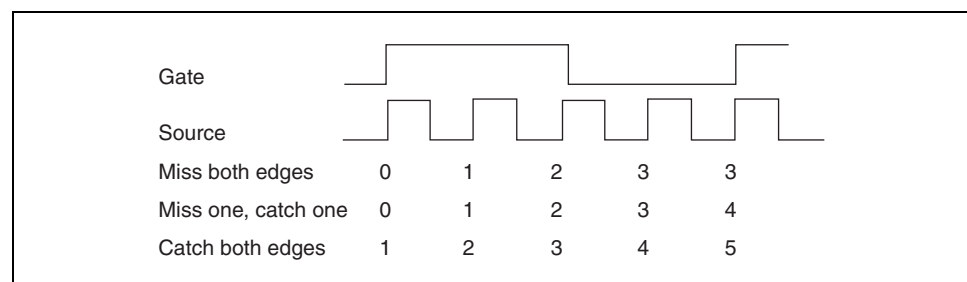
This section describes three ways to measure the frequency of a TTL pulse train using one or more counters. The frequency of a waveform is simply the inverse of its period at any given time. Thus, the easiest type of frequency measurement is merely the inverse of a period measurement. The two other types of frequency measurements become necessary because the first type of frequency measurement becomes increasingly inaccurate as the gate frequency approaches the timebase frequency of the counter.

Period

The first frequency measurement is really a period measurement. Once the period is acquired, take the inverse to compute frequency. The advantage to this method is that it uses only one counter and is easy to perform. However, this method relies on a relatively slow gate signal because the accuracy of a period measurement depends on the number of source edges that occur within one gate period.

Synchronization Error

As the gate frequency approaches the source frequency, period measurements suffer from synchronization error. For example, consider a period measurement that uses a 20 MHz internal timebase on the source. Now, suppose the gate signal is roughly 5 MHz, or 1/4 of the source frequency. The following illustration shows three possible scenarios in which the first and last source edges may or may not be included in the period measurement.



In the first scenario, the measurement misses the first and last source edges, counting a total of only three edges. The second scenario catches the first four edges and misses the last edge. The third scenario shows all five source edges being counted. The second scenario is obviously more accurate, but because the source edges are so closely synchronized with the gate edges, the counter is equally likely to pick any one of the three scenarios shown.

Pulse measurements always have an error of ± 1 source cycle, which is generally negligible when one source cycle accounts for only 1% (or less) of the pulse measurement. However, one source cycle accounts for between

33% to 20% of the measurement. This is known as synchronization error and can be avoided by choosing a different measurement scheme.

The following table shows how two frequencies, 50 kHz and 5 MHz, affect a period measurement.

Actual Frequency	Number of 50 ns Cycles	Measurement Error or +1 Cycle	Measurement Error of -1 Cycle	Frequency with Error of +1 Cycle	Frequency with Error of -1 Cycle
50 kHz	400	401	399	49.88 kHz	50.13 kHz
5 MHz	4	5	3	4 MHz	6.67 MHz

In NI-DAQmx, this method is called **Low Frequency with 1 Counter**.

Averaging

The second method of measuring frequency uses two counters—one to generate a pulse train with a known frequency and one to perform a period measurement. Counter 1 performs a period measurement, using the external signal as a source instead of the internal timebase. The gate signal of counter 1 comes from the output of counter 0, which is generating a pulse train. Because you know the frequency of the output of counter 0, you know exactly the length of the gate cycle on counter 1. Based on the number of source edges that arrive on the source of counter 1, you can deduce frequency, dividing the period measurement of counter 1 by the gate period. For example, if counter 0 outputs a pulse train of 10 Hz, the gate period is 0.1 s. If during that time, you count 100 source edges, you know the source frequency on counter 1 is $(100 \pm 1)/0.1$ or 1000 ± 10 Hz. In NI-DAQmx, this method is called **High Frequency with 2 Counters**.

Divide Down Method

The third method of measuring frequency also uses two counters except the counter that generates the pulse train (counter 0) uses the external signal as the source, and the counter that performs the period measurement (counter 1) uses the internal timebase as its source. Like the averaging method, the divide down method uses the pulse train from the output of counter 0 to gate the period measurement on counter 1.

The advantage to the divide down method is that it introduces less error than period measurements or averaging.

For example, suppose you program counter 0 for pulse train generation with pulse specs 5 and 5. This means that the delay and width are each made up of 5 periods of the source signal, and that the period of the resulting pulse train consists of 10 periods of the source signal (the source is divided down

by a factor of 10). In this example, counter 1 is configured for a period measurement, using the internal 20 MHz timebase as its source. If counter 1 registers 100 source edges during one gate period, you can deduce that the gate period lasted 5 μ s (50 ns \times 100 edges). You can therefore conclude that the external signal wired to the source of counter 0 had a period of 0.5 μ s or a frequency of 2 MHz.

Expressed as an equation,

$$F = (\text{pulse spec1} + \text{pulse spec2}) \times \text{timebase}/(\# \text{ of source edges} \pm 1)$$

In this example,

$$F = (5 + 5) \times 20,000,000/(100 \pm 1)$$

$$F = 200,000,000/101 \text{ to } 200,000,000/99$$

$$F = 1,980,198 \text{ to } 2,020,202 \text{ Hz}$$

In NI-DAQmx, this method is called **Large Range with 2 Counters**.

Exercise 9-7 Frequency Measurement

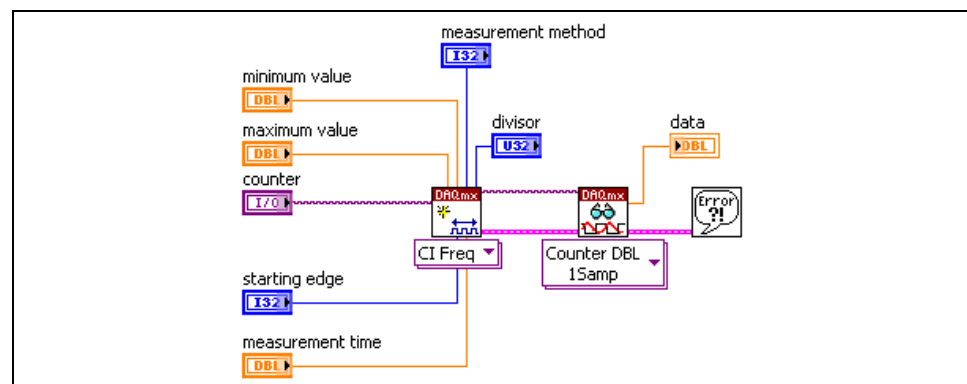
Objective: To build a VI that measures frequency using a counter.

This exercise explores three different methods of measuring frequency—inverse period, averaging, and divide down.

1. On the DAQ Signal Accessory, connect the square wave from the function generator to the gate of counter 1.
2. Open a blank VI and switch to the block diagram.

Inverse Period Method

3. Create the following block diagram.



- a. On the DAQmx Create Virtual Channel VI, right-click the **measurement method**, **minimum value**, **maximum value**, **counter**, **starting edge**, **measurement time**, and **divisor** inputs and select **Create»Control** from the shortcut menu.
 - b. On the DAQmx Read VI, select the **Counter»Single Sample»Double** instance from the pull-down menu. Right-click the **data** output and select **Create»Indicator** from the shortcut menu.
4. Save the VI as `Frequency Measurements.vi` in the `C:\Exercises\LabVIEW DAQ` directory.
 5. Set the controls on the front panel with the following values:
 - **Counter:** `DevX/ctrl1`, where `X` corresponds to the device number of your DAQ board
 - **Minimum value:** 2
 - **Maximum value:** 10,000
 - **Divisor:** 4
 - **Starting edge:** Rising
 - **Measurement time:** 1

6. On the DAQ Signal Accessory, set the Frequency Range to 100 Hz – 10 kHz. Turn the Frequency Adjust Knob as low as possible. The function generator should now produce a 100 Hz signal.
7. Run the VI. The data indicator should display approximately 100 Hz.
8. Experiment with the VI by adjusting the Frequency Adjust Knob and running the VI again.

Averaging Method

9. On the DAQ Signal Accessory, connect the square wave from the function generator to the source of counter 1.
10. Change the **measurement method** to High Frequency with 2 Counters. Decrease the **measurement time** to 0.001. Increase the **maximum value** to 1000000.
11. On the DAQ Signal Accessory, set the Frequency Range to 13 kHz – 1 MHz. Turn the Frequency Adjust Knob as high as possible.
12. Run the VI. The data indicator should display approximately 1 MHz.
13. Experiment with the VI by adjusting the Frequency Adjust Knob and running the VI again.

Divide Down Method

14. Change the **measurement method** to Large Range with 2 Counters. Set the **minimum value** to 5.
15. On the DAQ Signal Accessory, set the Frequency Range to 13 kHz – 1 MHz. Turn the Frequency Adjust Knob as high as possible.
16. Run the VI. The data indicator should display approximately 1 MHz.
17. Experiment with the VI by adjusting the Frequency Adjust Knob and running the VI again.

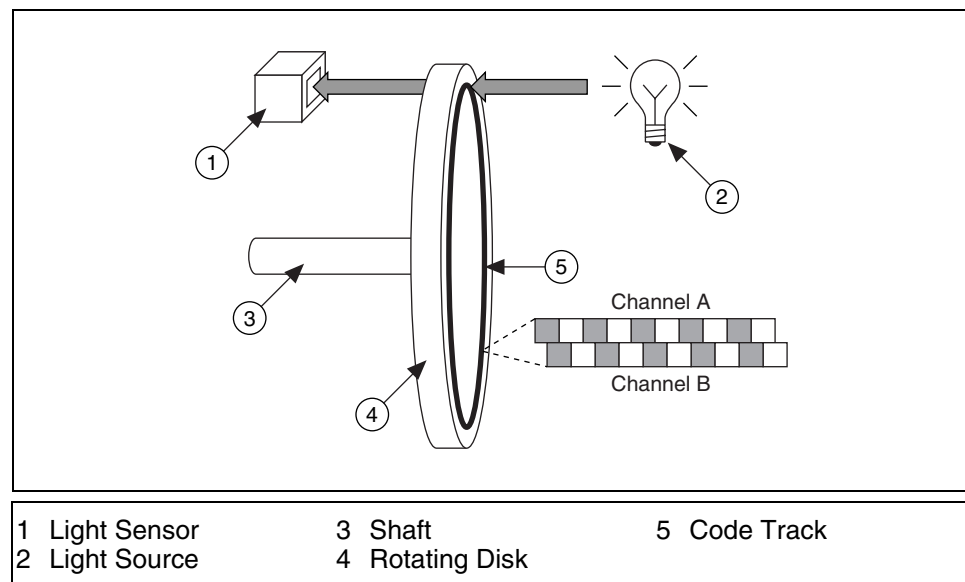
End of Exercise 9-7

I. Position Measurement

A quadrature encoder is a popular transducer used in counter applications. A quadrature encoder allows you to measure position, and it converts rotary motion into a measurable signal. The DAQ Signal Accessory has a quadrature encoder. Of the four counter chips, the NI-TIO is the only one that directly supports quadrature encoders. Quadrature encoders *can* be measured with both the Am9513 and the DAQ-STC, but neither chip was designed for encoder measurements. If you are measuring quadrature encoder signals, the best option is to use the NI-TIO chip.

How Encoders Work

An encoder is a transducer that allows you to measure position or distance. To understand how an encoder works, examine the quadrature encoder shown in the following illustration.

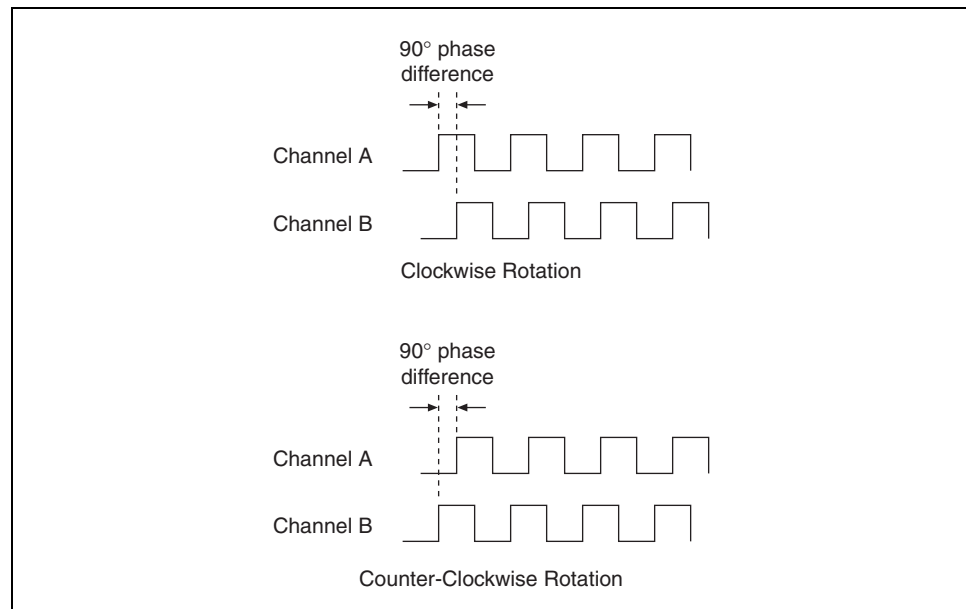


A quadrature encoder helps you convert rotary motion into a measurable signal. The rotary motion you are interested in is the spinning of the shaft. You could be interested in the direction the shaft is rotating, the speed the shaft is rotating, or both. A disk is attached to the shaft so that it rotates in the same direction and at the same speed as the shaft. The rotating disk is placed between a light source and a light sensor. The disk has a section of alternating opaque and transparent sections called the code track. An opaque section blocks the light from the source to the sensor, and a transparent section allows the light to pass from the source to the sensor. The code track consists of two rings of alternating opaque and transparent sections. Each ring produces a pulse train. The two rings are offset so that, depending on the direction the disk rotates, one pulse train leads the other. The number of opaque and transparent sections determines how many

pulses are produced per revolution. This is an important specification to know if you are keeping track of how many revolutions the shaft has completed. The quadrature encoder on the DAQ Signal Accessory produces 24 pulses per revolution.

Quadrature Encoder

Most encoders produce a TTL-compatible signal that can be used with a counter. As you saw earlier, a quadrature encoder produces two pulse trains. The two pulse trains are referred to as Channel A and Channel B. Channel A and Channel B will always be out of phase by 90 degrees, as shown in the following illustration. The leading channel is determined by the direction of rotation. If the encoder is rotated in a clockwise direction, Channel A will lead Channel B. If the encoder is rotated in a counter-clockwise direction Channel B will lead Channel A.



Note Some encoders might not produce a signal that is suitable for a counter. For example, some encoders produce a differential signal, and none of the four NI counter chips support differential inputs. If the signal produced by the encoder is not compatible with the counter chip you are using, you need to use signal conditioning before you send the signal to the counter.

Up/Down Line

To measure encoders with the DAQ-STC, you need to use a special input to the counter called the up/down line. The up/down line for counter 0 is the same pin as DIO6 on the pinout, and DIO7 for counter 1. The up/down line determines if an active edge on the source increments or decrements the count. If the signal sent to the up/down line is TTL high, an active edge on the source increments the count. If the signal sent to the up/down line is TTL low, an active edge on the source decrements the count.

DAQ-STC and Encoders

To measure an encoder with the DAQ-STC, you will use both the source and the up/down line. Connect Channel A to the source and Channel B to the up/down line. When you connect signals from the DAQ Signal Accessory, notice that Channel B is hardwired to DIO6, so the only connection that you need to make is from Channel A to the source of the counter you are using. Configure the counter for simple edge counting and set the active edge to falling. The count changes when a falling edge is received on Channel A, and Channel B determines if the count increments or decrements. When the encoder is turned clockwise, Channel A leads Channel B. When Channel A leads Channel B, the falling edge of Channel A occurs when Channel B is high, so the count increments. By the same logic, when the encoder is turned counter-clockwise, the count decrements. It is standard for clockwise motion to increment the count and counter-clockwise motion to decrement the count. If you would like to switch the two, you can set the active edge to rising instead of falling.

Refer to the *Using Quadrature Encoders with E Series DAQ Boards* Application Note for more information about using the DAQ-STC counter/timers to interface to quadrature encoders.

Exercise 9-8 Quadrature Encoder

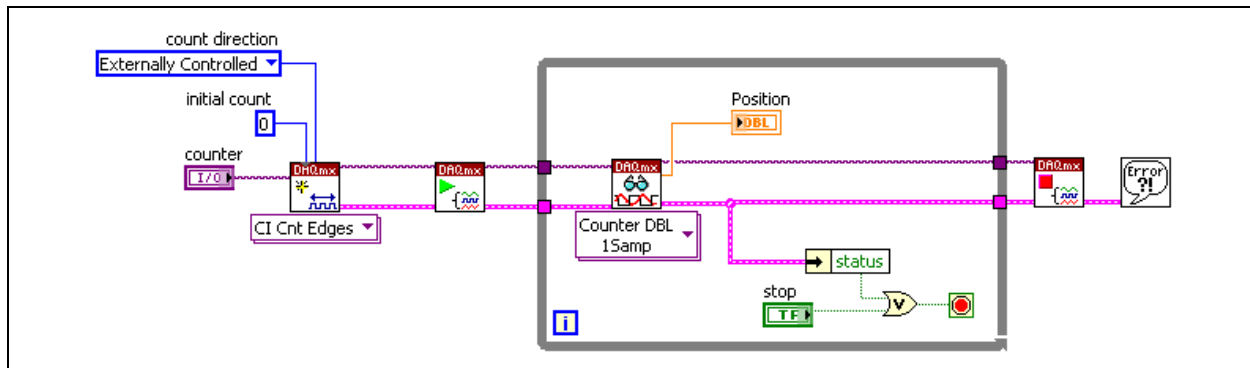
Objective: To measure the rotation of the quadrature encoder on the DAQ Signal Accessory by using Advanced Counter VIs.

1. On the DAQ Signal Accessory, connect Channel A from the quadrature encoder to the Source of Counter 0.



Note The up/down input for counter 0 is DIO6. Channel B on the DAQ Signal Accessory is hard-wired to DIO6. Therefore, no connection to the up/down input for counter 0 is necessary.

2. Open the Quadrature Encoder VI located in the `C:\Exercises\LabVIEW DAQ` folder.
3. Modify the block diagram as shown in the following figure.



The **count direction** terminal is specified as Externally Controlled. For STC-based devices, port0/line6 controls the count direction for counter 0.

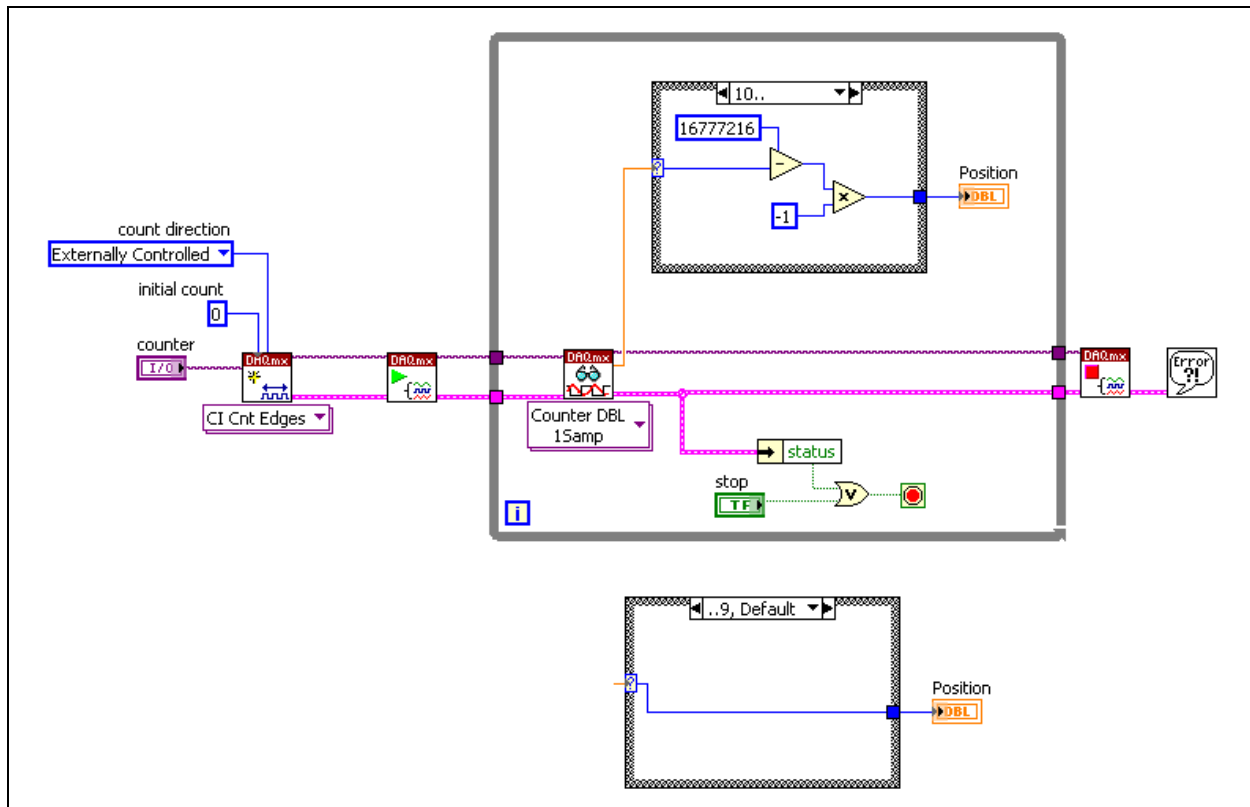
Front Panel

4. Return to the front panel and set the counter to `DevX/ctr0`, where X corresponds to the device number of your DAQ board.
5. Run the VI. Rotate the quadrature encoder knob on the DAQ Signal Accessory back and forth slowly to see how it affects the reading from the counter.

When you rotate the quadrature encoder knob in a clockwise direction, the position value increases, starting at zero. When you rotate the knob counterclockwise, the position value starts at zero and counts down from 16777215 ($=2^{24} - 1$). To translate this value to a more readable number, such as $-1, -2, \dots$, you have two options.

- You can read the state of port0/line6 to determine if the counter is counting up or down. If port0/line6 is in a count down state, you apply a 1's complement conversion to the position reading.
- You can impose a user defined limit for the maximum count. In this method, you ignore the state of port0/line6. When the value returned from the counter is greater than the user defined maximum count, you apply a 1's complement conversion to the position reading.

Modify the block diagram to implement the second option, as shown in the following figure.



In the Case structure, set the user defined maximum count at ten. When the position reading returned from the counter is greater than ten, convert the number to its 1's complement value. When the position reading is less than ten, leave the value unchanged. The value of ten is a small number, but is sufficient for demonstrating how to use this option.

6. Run the VI and rotate the quadrature encoder counter clockwise. Instead of seeing the very large numbers (in the 16,777,215 range), we now see the position reading relative to zero.
7. Save and close the VI.

End of Exercise 9-8

Summary

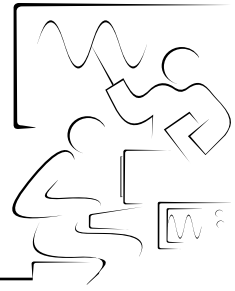
- Counters accept and generate TTL signals.
- The main components of a counter are the source, gate, out, and count register.
- NI devices can have one of two different counter chips:
 - DAQ-STC (24-bit)— E Series devices
 - NI-TIO (32-bit)—660x devices
- You can use the Easy Counter VIs to perform edge counting, pulse generation, pulse measurement, and frequency measurement.
- A quadrature encoder is a transducer that converts rotary motion into two pulse trains that are out of phase by 90 degrees.

Notes

Notes

Lesson 10

Synchronization



This lesson describes explicit state transitions, single device synchronization, and multiple device synchronization.

You Will Learn:

- A. Explicit State Transitions
- B. Single Device Synchronization
- C. Multiple Device Synchronization

A. Explicit State Transitions

NI-DAQmx uses a state model to control the resource allocation and execution flow of tasks. This state model is called the task state model. The task state model is very flexible and you can choose to interact with as little or as much of the task state model as your application requires. The DAQmx Start VI, DAQmx Stop VI, and DAQmx Control Task VI are used to transition the task from one state to another. You can explicitly transition between each task using the DAQmx Control Task VI, or you can allow NI-DAQmx to perform the state transitions implicitly. The task state model consists of five states—Unverified, Verified, Reserved, Committed, and Running.

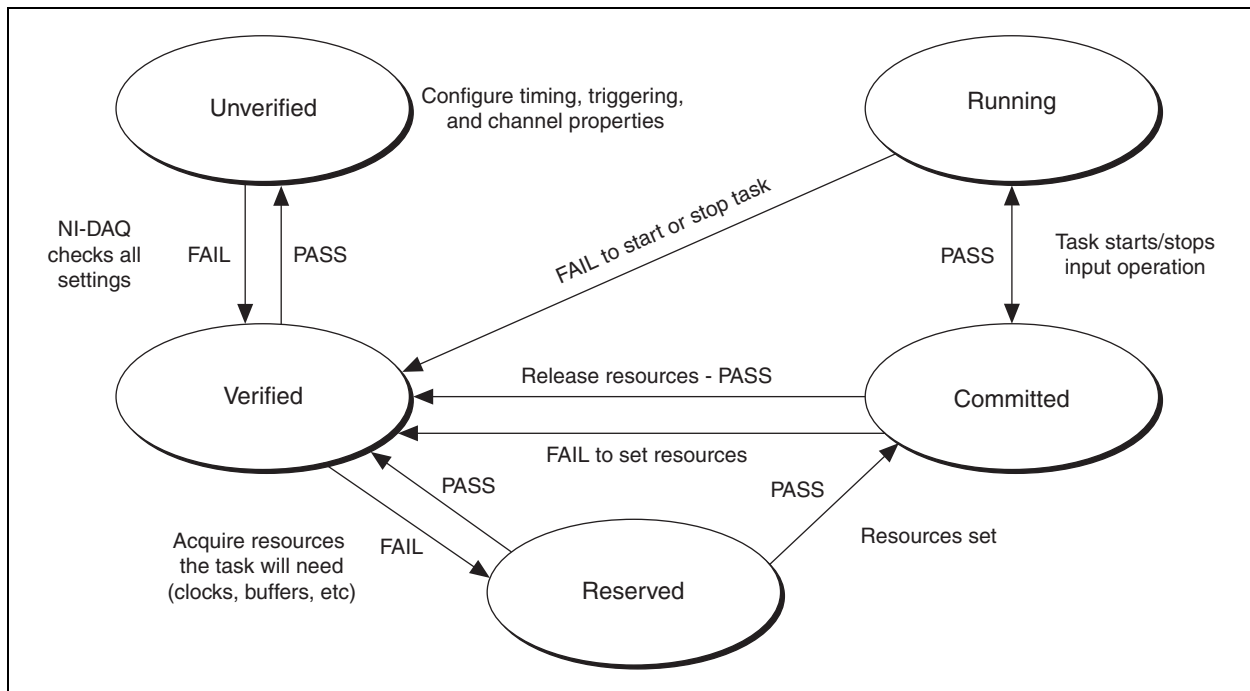
- **Unverified**—When a task is created or loaded, the default state is unverified. In this state, you configure the timing, triggering, and channel properties for the task.
- **Verified**—The timing, triggering, and channel properties that you set are checked, or verified, for their correctness when the task transitions from the unverified to the verified state. If all of the settings are valid, the task will successfully verify and move to the verified state. If any of the settings fail to verify, the task will remain in the unverified state. To explicitly perform this transition, call the DAQmx Control Task VI with the Action input set to Verify.
- **Reserved**—The resources a task uses to perform the specified operation are acquired exclusively when the task transitions from the verified to the reserved state. Example resources that need to be acquired include clocks or channels on a device or buffer memory in the computer. The resources must be reserved to prevent other tasks from using them and thus causing the first task to fail or encounter problems during operation. If the task successfully acquires all necessary resources, the task moves to the reserved state. Otherwise, the task remains in the verified state. To explicitly perform this transition, call the DAQmx Control Task VI with the Action input set to Reserve.
- **Committed**—After the necessary resources have been acquired, the settings for these resources must then be programmed. The successful programming of these settings causes the task to transition to the committed state. An example setting is the size of the buffer memory in the computer. If the state transition fails, the task is aborted and will return to the verified state. To explicitly perform this transition, call the DAQmx Control Task VI with the Action input set to Commit.

When the task resources for a specific operation are released, the task transitions from the committed state to the verified state. To explicitly perform this transition, call the DAQmx Control Task VI with the Action input set to Unreserve. When the task successfully releases all of its resources, it will transition back to the verified state.

- Running**—To transition from the committed state to the running state, the task must begin performing the specified operation. Calling the DAQmx Start VI invokes this transition. Starting a task does not necessarily start acquiring samples or generating waveforms. For example, timing and triggering properties might have been configured such that the sample is not acquired until you call the DAQmx Read VI or that a waveform is not generated until a trigger is enabled. If the transition to the running state fails, the task is aborted and returned to the verified state.

To stop the task from performing the specified operation, call the DAQmx Stop VI. This will transition the task from the running to the committed state. If this stop transition fails, the task is aborted and returned to the verified state.

The following illustration summarizes the task state model.



Explicit State Transitions

For the majority of data acquisition needs, you rarely need to explicitly interact with the Task State Model, and can rely on the task to perform implicit state transitions. However, there are instances when you should use explicit state transitions.

- Verify**—If the user of your application interactively configures a task by setting various channel, timing, and triggering properties, it may be beneficial to explicitly verify the task to inform the user if he or she has set a property to an invalid value.

- **Reserve**—If your application contains many different tasks that use the same set of resources, if one of these tasks repeatedly performs its operation, or if you want to ensure that none of the other tasks acquires these resources after the task begins its sequence of operations, then you should explicitly reserve a task.

Since reserving the task exclusively acquires the resources that are used by the task, this will ensure that other tasks cannot acquire these resources. For example, if your application contains two tasks that each perform a sequence of measurements and you want to ensure that each sequence is completed before the other sequence begins, this behavior can be enforced by explicitly reserving each task before it begins its sequence of measurements.

- **Commit**—It may be beneficial to explicitly commit a task if your application performs multiple measurements or generations by repeatedly starting and stopping a task. Committing the task exclusively acquires the resources that the task uses and programs some of the settings for these resources. By explicitly committing the task, these operations are performed once, not each time the task is started, which can considerably decrease the time needed to start your task. For example, if your application repeatedly performs finite, hardware-timed measurements, the time required to start the task may be dramatically decreased if you explicitly commit the task before repeatedly performing these measurements. Explicitly committing a task also is required if you need to perform additional read operations of the samples acquired by the task after stopping the task.
- **Start**—If your application repeatedly performs read or write operations, it may be beneficial for you to explicitly start a task. Starting the task reserves the resources that the task uses, programs some of the settings for these resources, and begins to perform the specified operation. These operations are performed once, not each the time read or write operation is performed, by explicitly starting the task. This can considerably decrease the time required to perform each read or write operation. For example, if your application repeatedly performs single-sample, software-timed read operations, the time required for each read operation may be dramatically decreased if you explicitly start the task before repeatedly performing these read operations.

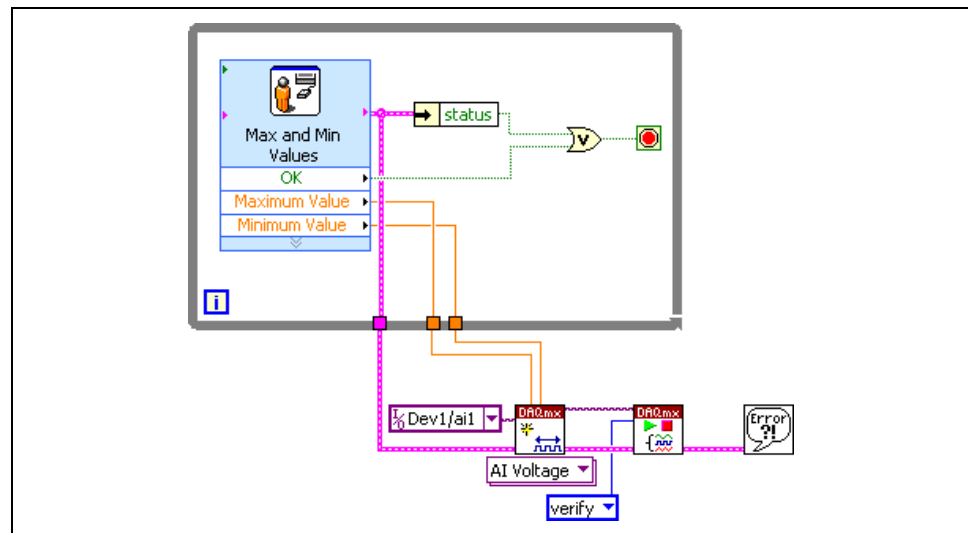
Exercise 10-1 Explicit State Transitions

Objective: To build a VI to explicitly transition between the various states of the NI-DAQmx Task State Model.

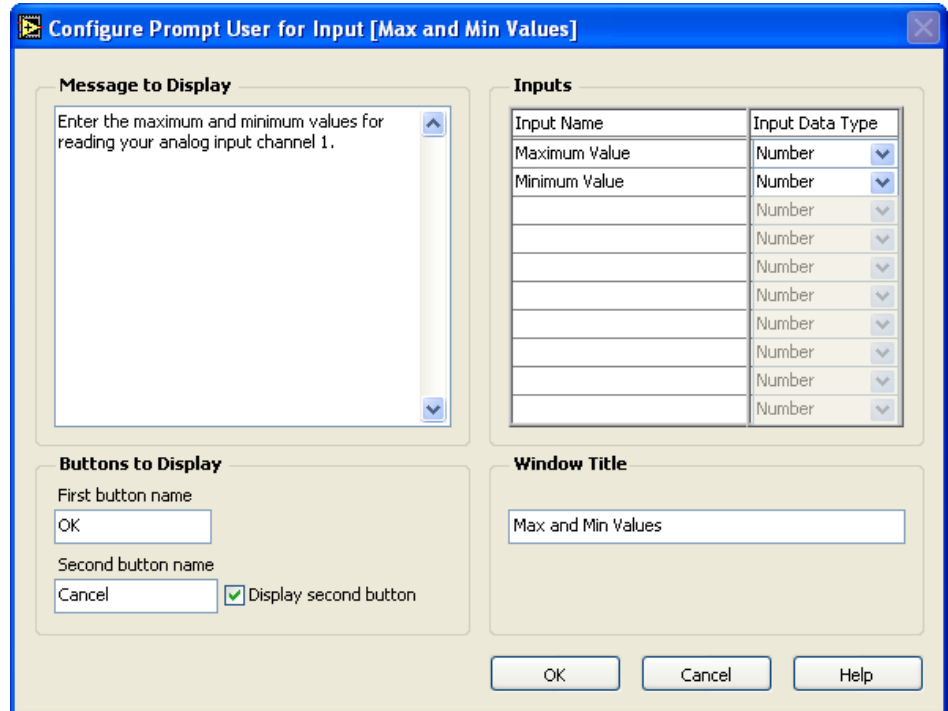
There are times in which you want to explicitly transition between tasks in the task state model. In this exercise, you explore the most common case in which this is necessary—user required input for creating virtual channels.

Block Diagram

1. Open a blank VI and build the following block diagram.



- a. Place the Prompt User for Input Express VI, located on the **Functions»Input** palette, on the block diagram. This Express VI creates a dialog box to display a custom message that prompts the user to input data. In the **Configure Prompt User for Input** dialog box that displays, use settings shown in the following example.



- b. Place the DAQmx Create Virtual Channel VI, located on the **Functions»All Functions»NI Measurements»DAQmx - Data Acquisition** palette, on the block diagram. This VI creates a new virtual channel. Select the **Analog Input»Voltage** instance from the pull-down menu. Right-click the **physical channel** input and select **Create»Constant** from the shortcut menu. Select $DevX/ai1$, where X corresponds to the device number of your DAQ device.



- c. Place the DAQmx Control Task VI, located on the **Functions»All Functions»NI Measurements»DAQmx - Data Acquisition»DAQmx Advanced Task Options** palette, on the block diagram. This VI alters the state of a task based on the **action** that you specify. Right-click the **action** input and select **Create»Constant** from the shortcut menu. Select **verify** from the pull-down menu to transition the task into the Verified state when all the configuration settings are correct.



- d. Place the Simple Error Handler VI, located on the **Functions»All Functions»Time & Dialog** palette, on the block diagram.
2. Run the VI. A dialog box displays to prompt you for the maximum and minimum values of the channel. Enter 5 as the maximum value and 10 as the minimum. Click the **OK** button.

B. Single Device Synchronization

Many applications require making more than one type of measurement at the same time. Simultaneous measurements involve different operations happening at the same time, such as acquiring data on input channels while generating data on output channels. However, these operations are not necessarily correlated to one another. For example, you can start an input operation at the same time you start an output operation but from there, each operation can run independently at its own rate.

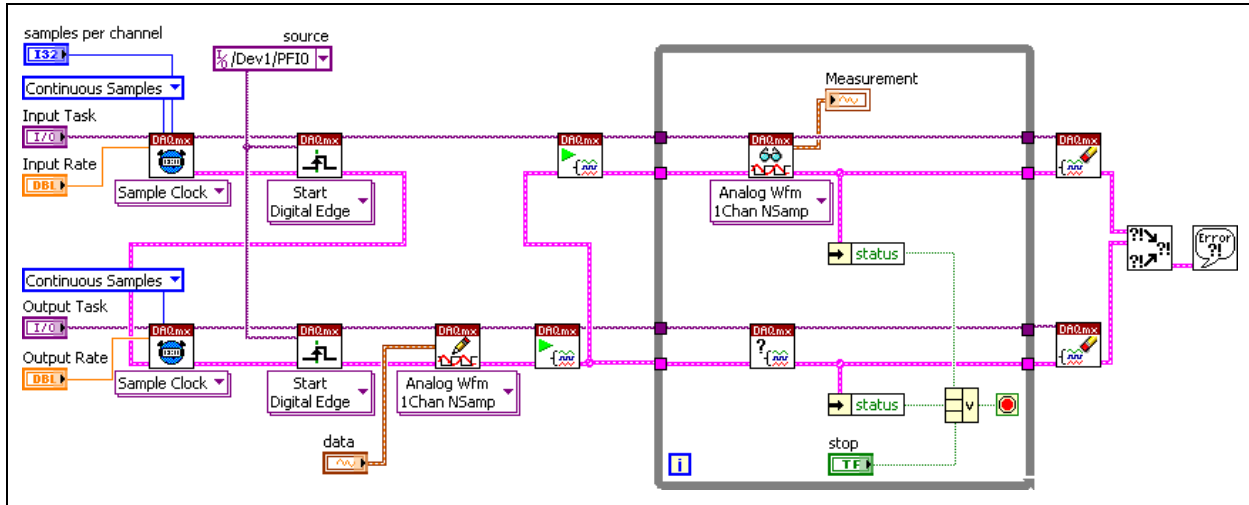
A synchronized measurement is when the measurements are all taken at the same time. When synchronizing measurements, such as acquiring one hundred temperature and speed measurements, you need to start all measurements at the same time. In addition, the measurements must share a common clock for latching data.

For example, in control-loop applications, you need to make multiple measurements at the beginning of the loop, perform a calculation based on the new measurements, and finally output data based on this calculation. This type of application requires that you start all the measurements at the same time and synchronize them via a common clock signal. Similarly, if you want to correlate measurements, such as plotting speed and brake-pad temperature versus time, you will need to first synchronize both the speed and temperature measurements.

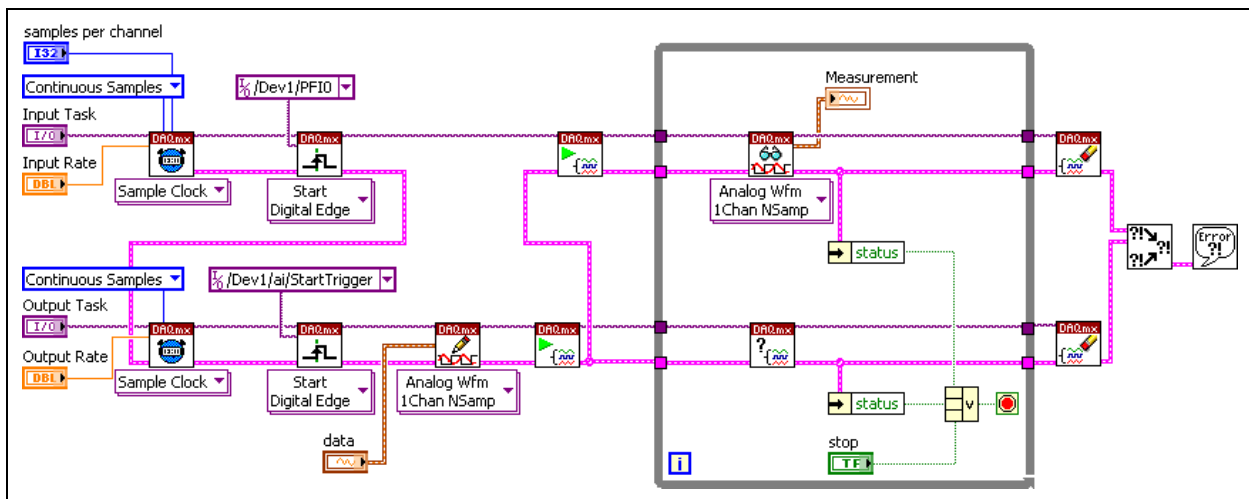
Simultaneously Started Measurements

To simultaneously start an analog input and output operation, trigger the operation via a hardware trigger or a software trigger. For a hardware triggered operation, both the analog input and output operation trigger off of the same PFI or RTSI pin. Refer to the *Routing Signals and RTSI* section for more information about RTSI.

The following block diagram demonstrates how to perform this type of hardware triggered operation.

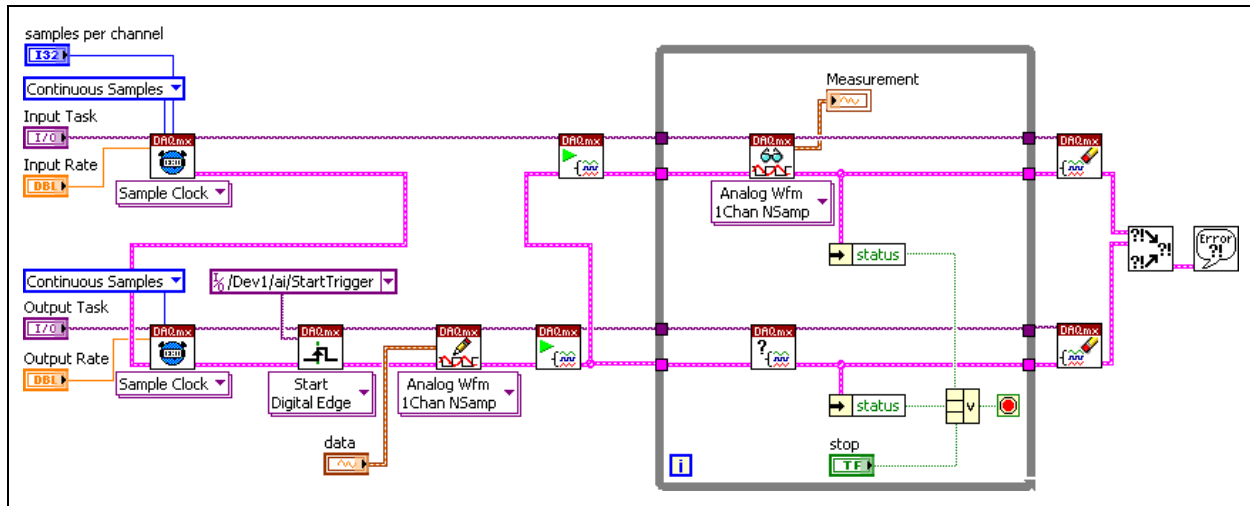


For a software triggered operation, the analog input is triggered off of an external PFI or RTSI pin, and the analog output triggers off the internal AI Start Trigger signal. AI Start Trigger is an internal signal that is directly connected to both the analog input and analog output subsystems. The software triggered method is slightly more accurate than the hardware triggered method because the external signal has only to propagate through one main path to reach both subsystems. However, this delay is almost always insignificant at the speeds at which MIO devices operate. The following diagram illustrates this technique.



The above example is configured so the input operation responds to a digital start trigger on PFI0, configured with the DAQmx Trigger VI. The output operation uses the DAQmx Trigger VI to trigger off of the internal AI Start Trigger. Notice that the output operation must be started before the input operation to ensure that the input does not start and send the internal AI Start Trigger before the output operation is set up to receive the start signal.

Another example of a simultaneous start based on a software trigger occurs when the analog input is started with a software call instead of a hardware trigger. The analog output is still called using the internal AI Start Trigger signal. The following diagram illustrates this fully software triggered example.



Once the operations are simultaneously started, they are not necessarily synchronized. The operations can be set to acquire and generate data independently of another at different rates.

Exercise 10-2 Simultaneously Started Analog Input and Output

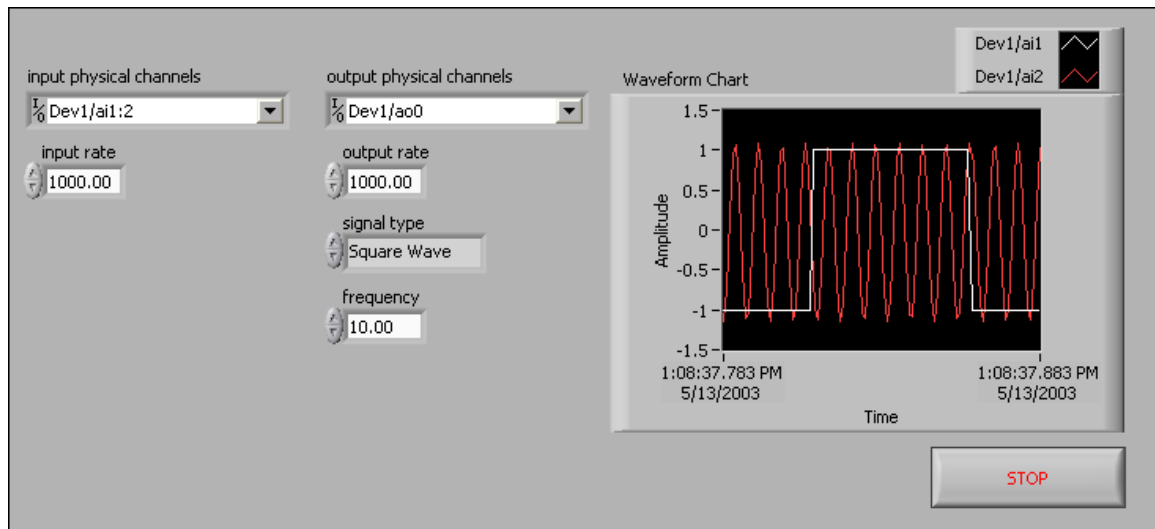
Objective: To use two different methods to simultaneously start an analog input and output operation.

The first method in this exercise uses a hardware trigger to begin the analog input operation and the second method uses a software call to start the analog input. Both methods use the DAQmx Timing VI to trigger the analog output off of the internal AI Start Trigger signal.

Method I – Hardware Triggered

Front Panel

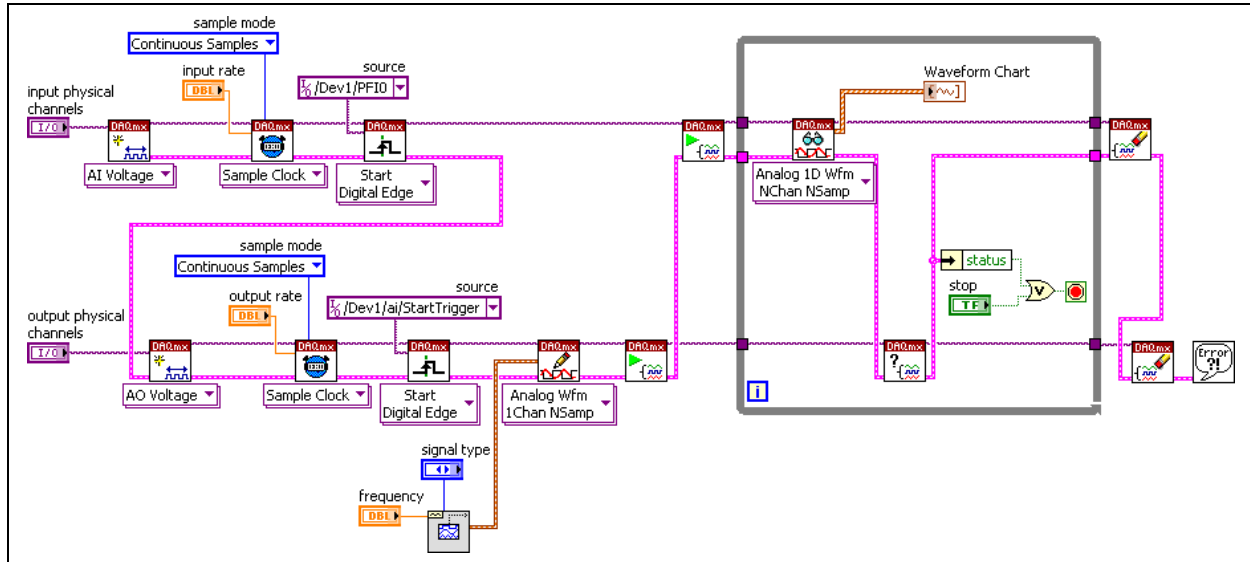
1. Open a blank VI and build the following front panel.



Do not create the **signal type** control on the front panel. Create this control from the block diagram.

Block Diagram

2. Build the following block diagram.



- a. Place the DAQmx Create Virtual Channel VI, located on the **Functions»All Functions»NI Measurements»DAQmx - Data Acquisition** palette, on the block diagram. This VI creates a new virtual channel. Use two of these VIs. Select the **Analog Input»Voltage** instance from the pull-down menu for the first use of this VI. Select the **Analog Output»Voltage** instance from the pull-down menu for the second use of this VI.



- b. Place the DAQmx Timing VI, located on the **Functions»All Functions»NI Measurements»DAQmx - Data Acquisition** palette, on the block diagram. This VI sets the task timing. You use two of these VIs. For both uses of this VI, set the **sampling mode** to **Continuous**.



- c. Place the DAQmx Trigger VI, located on the **Functions»All Functions»NI Measurements»DAQmx - Data Acquisition** palette, on the block diagram. This VI sets the trigger configuration for a task. For the analog input task, set the source as `/DevX/PFI0`. For the analog output task, set the source as `/DevX/ai/StartTrigger`. Where `X` corresponds to the device number of your DAQ device in both cases.



- d. Place the Basic Function Generator VI, located on the **Functions»All Functions»Waveform»Analog Waveform»Waveform Generation** palette, on the block diagram. This VI creates a waveform based on the **signal type**. Right-click the **signal type** input and select **Create»Control** from the shortcut menu.



- e. Place the DAQmx Write VI, located on the **Functions»All Functions»NI Measurements»DAQmx - Data Acquisition** palette, on the block diagram. Select the **Analog»Single Channel»Multiple Samples»Waveform** instance from the pull-down menu.



f. Place the DAQmx Start Task VI, located on the **Functions»All Functions»NI Measurements»DAQmx - Data Acquisition** palette, on the block diagram. This VI starts a task.



g. Place the DAQmx Read VI, located on the **Functions»All Functions»NI Measurements»DAQmx - Data Acquisition** palette, on the block diagram. Select the **Analog»Multiple Channels»Multiple Samples»Waveform** instance from the pull-down menu.



h. Place the DAQmx Is Task Done VI, located on the **Functions»All Functions»NI Measurements»DAQmx - Data Acquisition»DAQmx Advanced Task Options** palette, on the block diagram. This VI queries the status of the task and returns if it completes execution. Using this VI to poll the analog output task for any error that might occur.



i. Place the DAQmx Clear Task VI, located on the **Functions»All Functions»NI Measurements»DAQmx - Data Acquisition** palette, on the block diagram. This VI stops the task and clears all resources.



j. Place the Unbundle by Name function, located on the **Functions»All Functions»Cluster** palette, on the block diagram.



k. Place the Or function, located on the **Functions»Arithmetic & Comparison»Boolean** palette, on the block diagram.



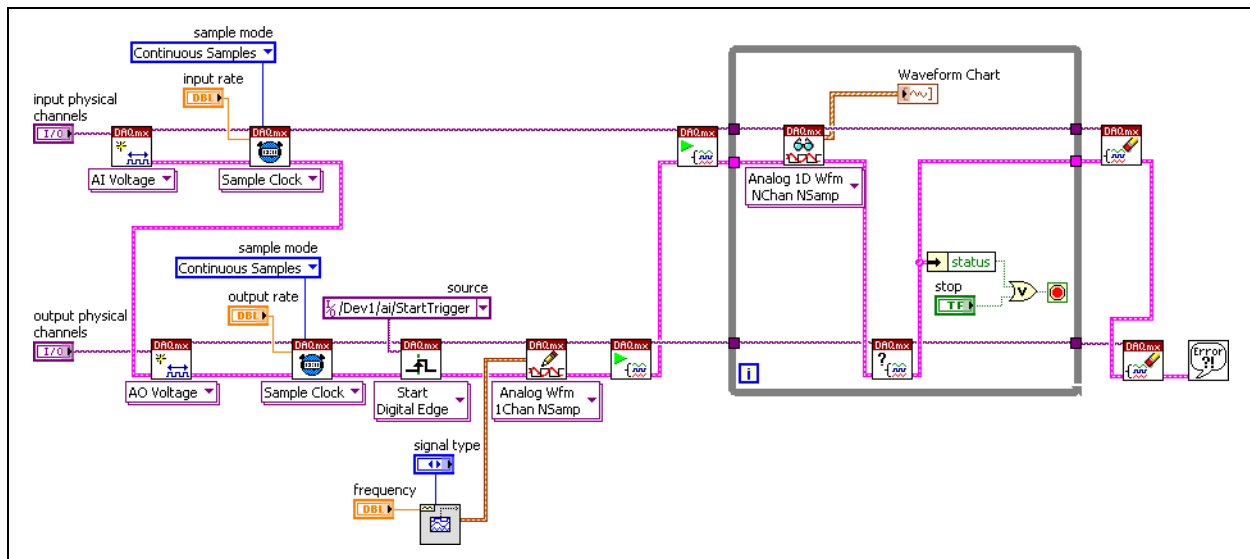
l. Place the Simple Error Handler VI, located on the **Functions»All Functions»Time & Dialog** palette, on the block diagram.

3. Save the VI as *Simultaneous AI AO Start - HW.vi* in the `C:\Exercises\LabVIEW DAQ` directory.
4. On the DAQ Signal Accessory, connect analog out 0 to analog in 1. Connect the sine wave on the function generator to analog in 2.
5. On the front panel, set the controls with the following values.
 - **Input physical channels:** `DevX/ai1:2`, where *X* corresponds to the device number of your DAQ device
 - **Input rate:** 1000
 - **Output physical channels:** `DevX/ao0`, where *X* corresponds to the device number of your DAQ device
 - **Output rate:** 1000
 - **Signal type:** Sine Wave
 - **Frequency:** 100
6. Run the VI. Press the Digital Trigger button on the DAQ Signal Accessory to start the operations.
7. Stop the VI.

8. Modify the waveform chart to display the data in a more friendly fashion.
9. Select a different **signal type**. Run the VI. Press the Digital Trigger button on the DAQ Signal Accessory to start the measurement.
10. Stop the VI, but do not close it.

Method II – Software Triggered

1. Select **File»Save As** and save the VI from Method I as Simultaneous AI AO Start - SW.vi in the C:\Exercises\LabVIEW DAQ directory.
2. Modify the block diagram as shown in the following figure.



Remove the DAQmx Trigger VI from the analog input task. Instead of using a hardware trigger to begin the input measurement, the measurement begins when you click the Run button. Analog output continues to trigger off the start of the analog input.

3. On the DAQ Signal Accessory, connect analog out 0 to analog in 1. Connect the sine wave on the function generator to analog in 2.
4. On the front panel, set the controls with the following values.
 - **Input physical channels:** DevX/ai1:2, where X corresponds to the device number of your DAQ device
 - **Input rate:** 1000
 - **Output physical channels:** DevX/ao0, where X corresponds to the device number of your DAQ device
 - **Output rate:** 1000
 - **Signal type:** Sine Wave
 - **Frequency:** 100

5. Run the VI.
6. Stop the VI and select a different **signal type**. Run the VI.
7. Stop the VI. Save and close all VIs.

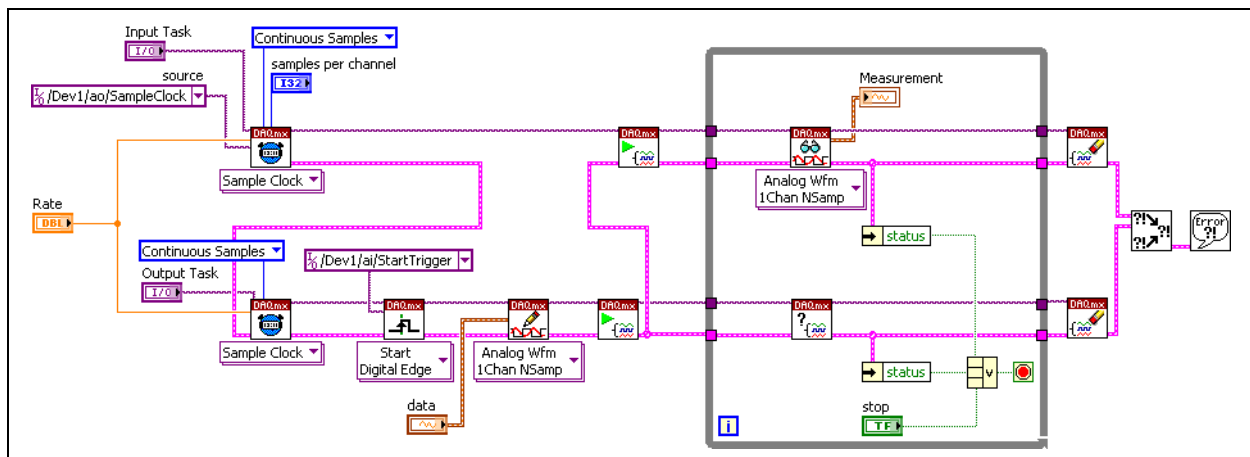
End of Exercise 10-2

Synchronized Measurements

When you want to fully synchronize the analog input and output operations such that each input sample per channel occurs at the same time each output sample per channel is updated, the operations must use a common clock source. Like simultaneous starting operations, there are two main methods used to synchronize methods.

The first method uses the internal AI Start Trigger to trigger the analog output to start at the same time as the input acquisition. Unlike the simultaneous start example, we now set the input and output sample clocks to run at the same rate, causing the operations to be synchronized on a single device. Both the internal input and output sample clocks are derived from the onboard timebase of the device. Since both clocks are derived from the same timebase and started at the same time, the sample clocks will be synchronized. Because the analog input and output subsystems each have their own divide down circuitry to derive their own sample clocks, there could be a small phase difference between the two sample clocks. However, these differences are insignificant with the rates at which E Series devices run.

Another method to synchronize analog input and output is to have both operations use either the analog input or analog output sample clock. The following diagram demonstrates how to synchronize analog input and output by sharing the analog output sample clock.



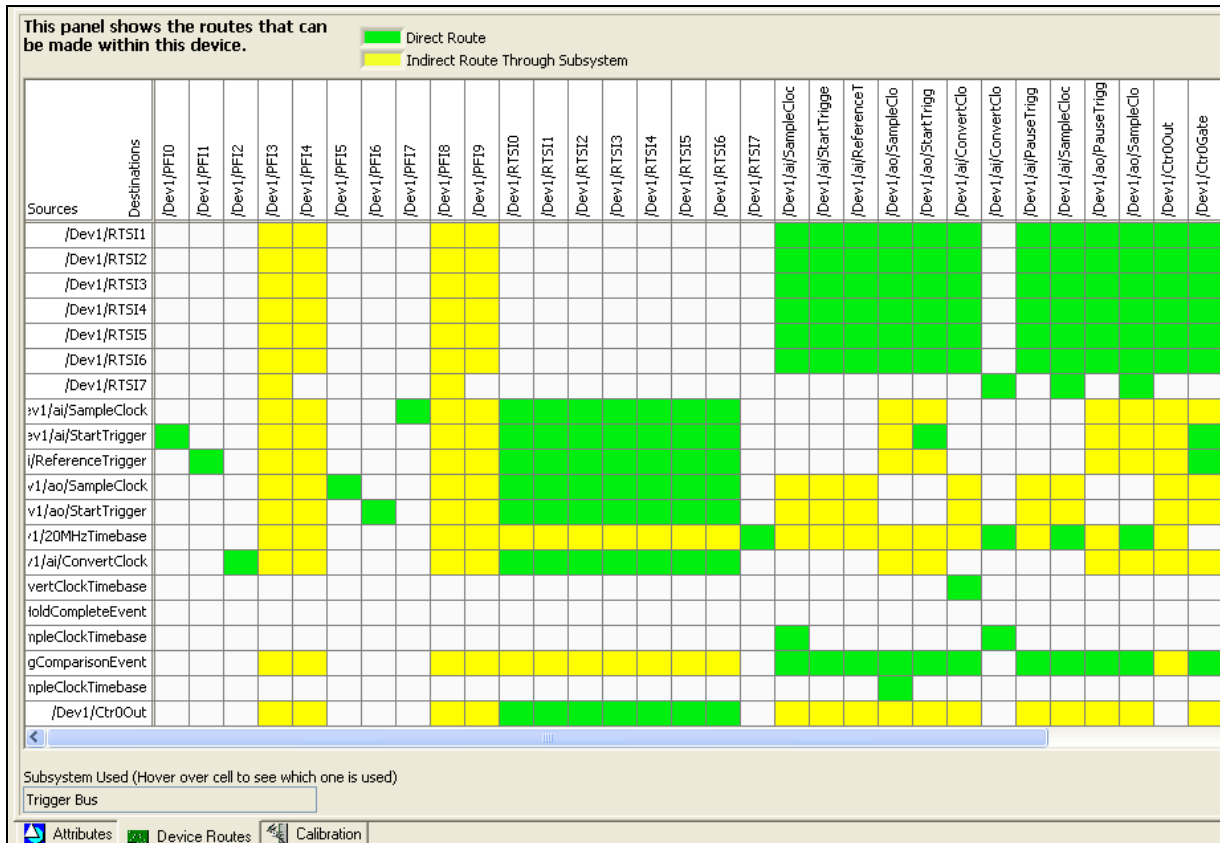
To perform a synchronized analog input and output operation with a hardware start trigger, use one of the synchronization methods previously discussed and simply add a hardware start trigger to the master operation, which in all of the previous examples has been the analog input operation.

Routing Signals and RTSI

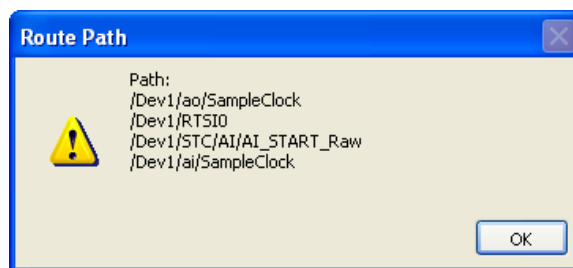
Some internal paths exist between subsystems of a device that allow certain signals to be internally routed between these subsystems. We have already discussed how the internal AI Start Trigger signal is directly connected to both the analog input and output subsystems of an E Series device. Other examples of signals that can be routed internally on an E Series device include the output of counter 0 to the AI Sample Clock, AI Start Trigger, or AI Reference Trigger. Also, the output of counter 1 can be directly routed to the AO Sample Clock. Refer to the *NI-DAQmx Help* for a complete list of routable signals. All other signal routes must be made externally through a PFI or RTSI pin.

NI-DAQmx manages the majority of routing between different subsystems for you. Instead of having to route the analog output clock output to a PFI pin to be used by the analog input task, NI-DAQmx will allow you to just specify the AO Sample Clock as the AI Sample Clock source. The routing is done behind the scenes for you.

In MAX, select your NI-DAQmx device under the **NI-DAQmx Devices** section. The routing table in the **Device Routes** tab shows the possible routes that can be made within the device—both directly and indirectly. Routes that can be made directly, such as the internal route between the output of counter 0 and the AI Sample Clock, are marked in green. Routes that can only be made between two terminals by going through RTSI lines or other subsystems are shown in yellow. When a route is made by the driver, all of the terminals involved in the route are reserved for that route. If you hold down the <Shift> key and right-click a route by right-clicking the yellow or green square, the path that makes up the route is displayed. The following diagram is part of the routing table for a PCI-MIO-16E-4 device.



The following figure shows the path between the AO Sample Clock and the AI Sample Clock.



Real-Time System Integration (RTSI) Bus

RTSI is an internal timing bus used to share and exchange timing and control signals between multiple boards through the use of parallel digital lines. The connector is typically located on the top of a DAQ device. The advantage of RTSI is that it allows you to programmatically pass digital signals to use for triggering, clocking, and other tasks between multiple DAQ devices or PXI modules. For synchronization applications, the RTSI bus can be used to allow one board to generate the clock and trigger signal and pass those signals over the bus.

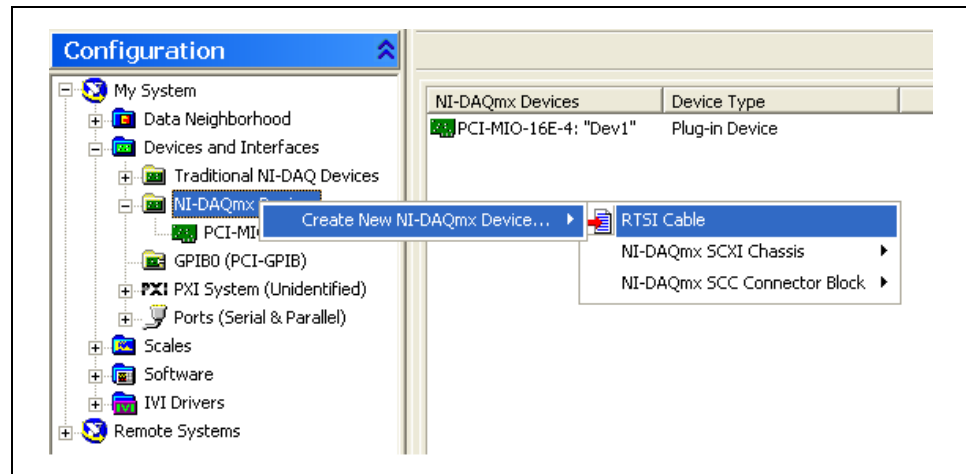
To register your RTSI cable in MAX, right-click **NI-DAQmx Devices**, select **Create New NI-DAQmx Device** from the shortcut menu, and select **RTSI Cable**. A RTSI cable should appear under your NI-DAQmx devices. Right-click the RTSI cable and add the devices it is connected to.

Exercise 10-3 RTSI Configuration

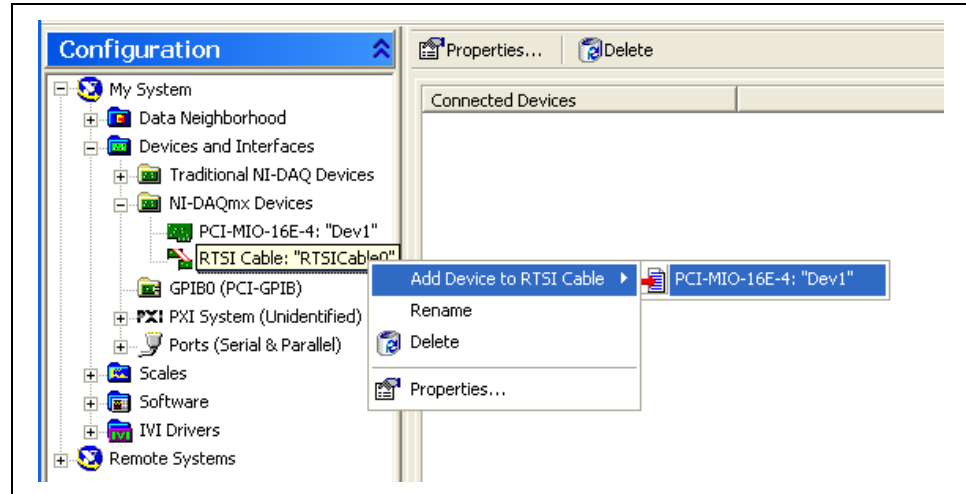
Objective: To configure a RTSI cable in MAX.

A RTSI cable is most often used to transfer signals from one DAQ device to another. Later in this lesson, you will learn more about multiple device synchronization. However, certain signals require a RTSI cable to be configured and assigned to a device in order to transfer the signal on a single DAQ device. Two of these types of signals are the AO Sample Clock and AI Sample Clock, which you will use in Exercise 10-4 to synchronize an analog input and analog output task. In preparation for that exercise, you will first configure a RTSI cable in MAX.

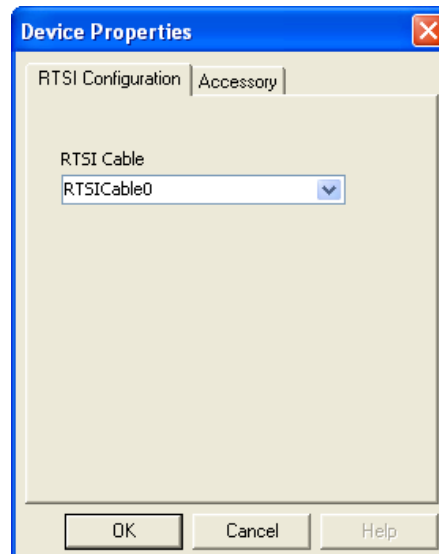
1. Launch MAX by double-clicking the icon on the desktop or by selecting **Tools»Measurement & Automation Explorer** in LabVIEW.
2. Under Devices and Interfaces, right-click **NI-DAQmx Devices**.
3. Select **Create New NI-DAQmx Device»RTSI Cable** from the shortcut menu.



4. The RTSI Cable should now appear under NI-DAQmx Devices as **RTSI Cable: "RTSICable0"**. Right-click this listing and select **Add Device to RTSI Cable** from the shortcut menu. Your DAQ device should be listed. Select your DAQ device.



You also can right-click your NI-DAQmx device, select **Properties** from the shortcut menu, and select the RTSI cable in the **RTSI Configuration** tab of the **Device Properties** dialog box, as shown in the following figure.



End of Exercise 10-3

Exercise 10-4 Synchronized Analog Input and Output

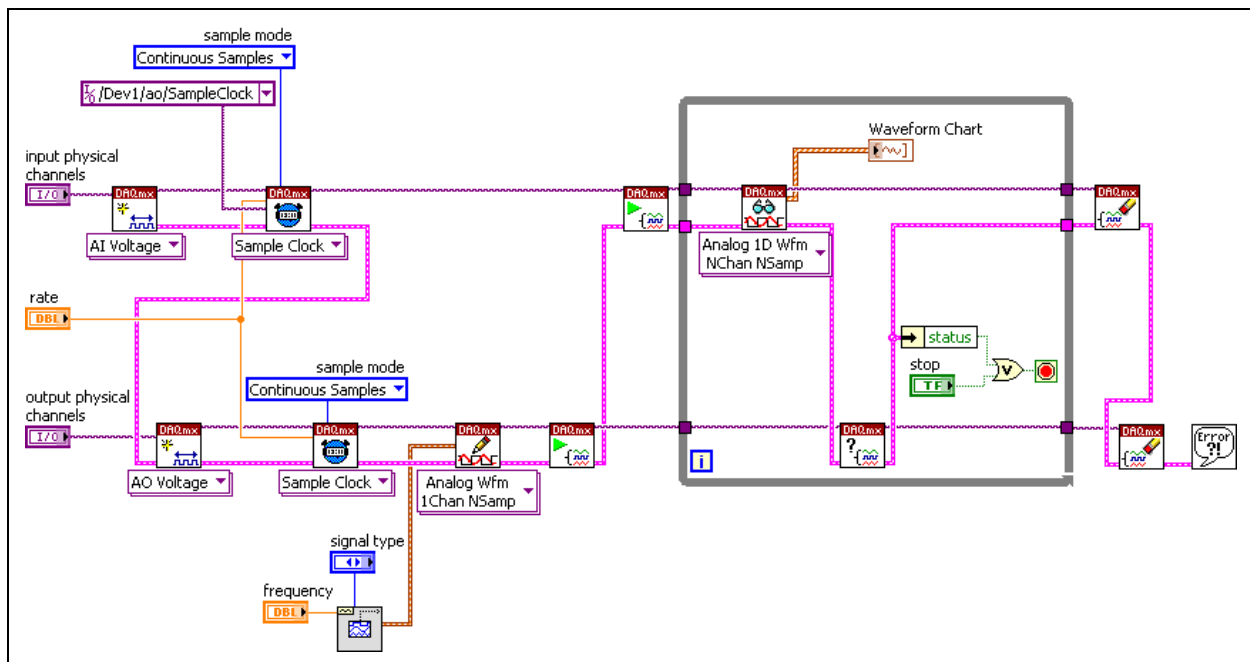
Objective: To perform a synchronized analog input and output operation by sharing the AI Sample Clock.

To synchronize the generation and measurement of analog signals, share either the AO Sample Clock or the AI Sample Clock between the analog input and analog output tasks.

1. Open the Simultaneous AI AO Start - SW VI located in the C:\Exercises\LabVIEW DAQ directory.
2. Select **File>Save As** and save the VI as Synchronized AI and AO.vi in the C:\Exercises\LabVIEW DAQ directory.

Block Diagram

3. Modify the block diagram as shown in the following figure.



Notice that the **rate** must be shared between the analog input and analog output task in order to synchronize the timing between the two tasks. In addition, set the **source** of the analog input timing to /DevX/ao/SampleClock, where X corresponds to the device number of your DAQ device.

4. On the DAQ Signal Accessory, connect analog out 0 to analog in 1. Connect the sine wave on the function generator to analog in 2.

5. Switch to the front panel and set the controls with the following values:
 - **Input physical channels:** DevX/ai1:2, where *X* corresponds to the device number of your DAQ device
 - **Output physical channels:** DevX/ao0, where *X* corresponds to the device number of your DAQ device
 - **Rate:** 10000
 - **Signal Type:** Square Wave
 - **Frequency:** 100
6. Run the VI.
7. Stop the VI. Save and close the VI.

End of Exercise 10-4

Using External Counters

There are several applications in which it is necessary to perform counter operations simultaneously with analog input and/or output operations. Some of the common cases involve using a counter to provide an external sample clock, using a counter to perform retriggerable analog operations, using a counter to start an analog operation after a certain number of triggers occur and to correlate event counting measurements to analog operation.

Analog Input with the External Clock Generated by a Counter

Use a counter to generate a pulse train, either finite or continuous, to serve as the sample clock for analog input or output operations. The frequency of a pulse train being generated by a counter can be changed on-the-fly, allowing you to correspondingly change the rate of an analog input or output operation. Configure a counter for continuous pulse train generation. Set the sample clock source for the analog input acquisition to the internal output of the counter.



Note The frequency of the AO Sample Clock can be changed on-the-fly. Therefore, you do not need to use a counter to change the rate of an analog output generation at run time.

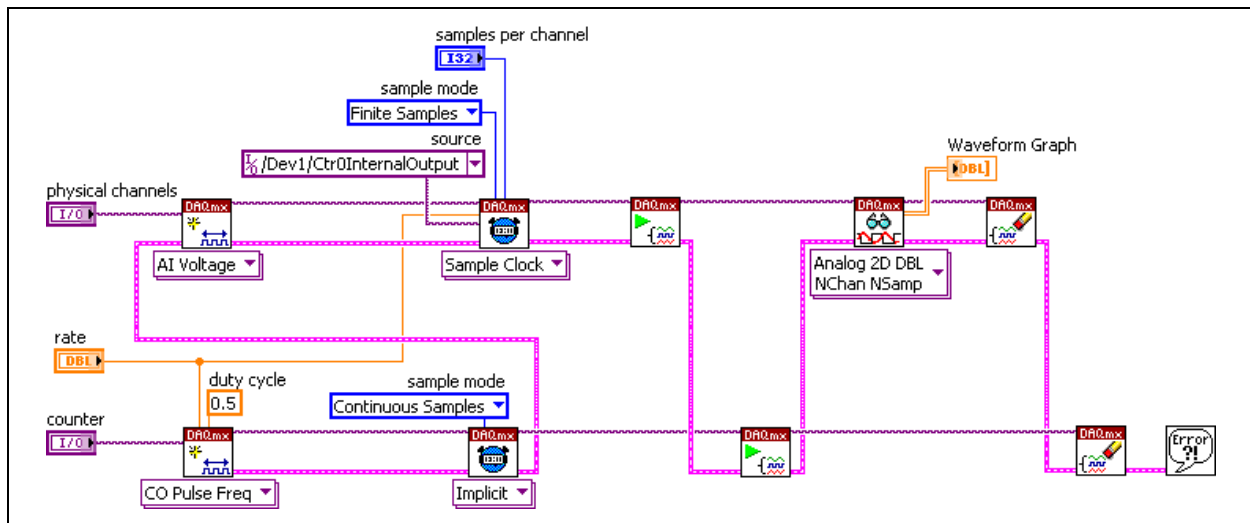
Exercise 10-5 Analog Input – External Clock Generated by Counter

Objective: To build a VI that uses a counter to generate a pulse train to clock an analog input acquisition.

In this exercise, a counter creates a pulse train that acts as the analog input sample clock. Because you can change the pulse train frequency of the counter on-the-fly, you can change the analog input sample clock at run time.

Block Diagram

1. Open a blank VI and build the following block diagram.



After you create the constant on the **source** input of the DAQmx Timing VI, right-click it and select **I/O Name Filtering** from the shortcut menu. Place a checkmark in the **Include Advanced Terminals** checkbox and click the **OK** button to close the **Filter Names** dialog box. This allows you to select `/DevX/Ctr0InternalOutput`, where `X` corresponds to the device number of your DAQ device.

2. Save the VI as `AI - External Clock from Counter.vi` in the `C:\Exercises\LabVIEW DAQ` directory.
3. Connect the sine wave on the function generator of the DAQ Signal Accessory to analog in 1.

4. On the front panel, set the controls as follows:
 - **Physical Channels:** DevX/ai1, where X corresponds to the device number of your DAQ device
 - **Counter:** DevX/ctr0, where X corresponds to the device number of your DAQ device
 - **Rate:** 1000
 - **Samples per channel:** 200
5. Run the VI.
6. Increase the rate and run the VI. This increases the number of samples acquired each second.
7. Save and close the VI.

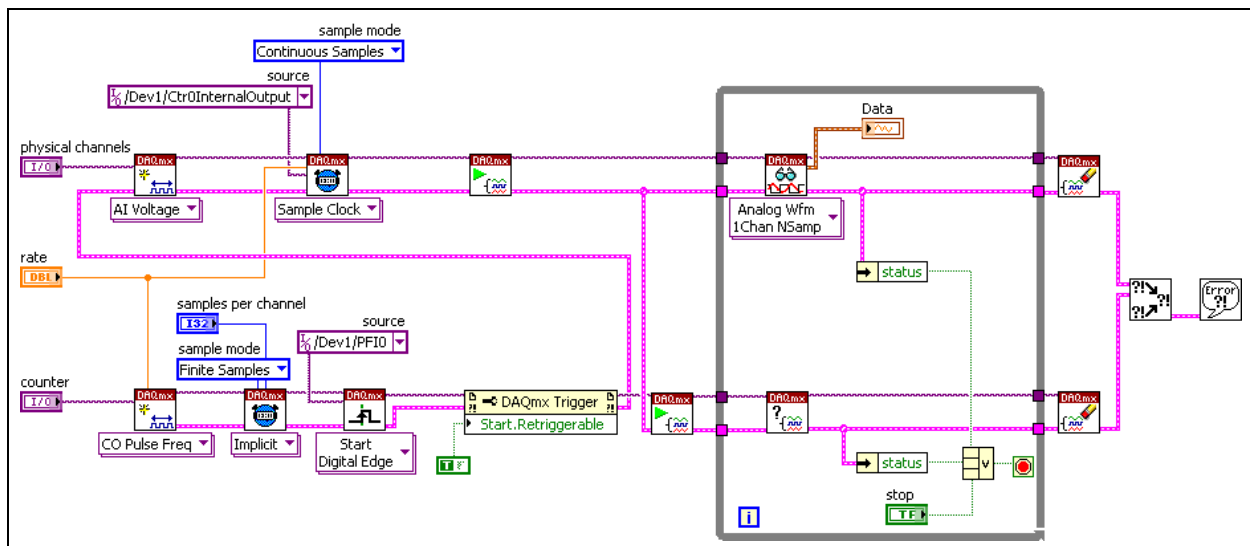
End of Exercise 10-5

Retriggerable Analog Operations

Analog input and output operations are not hardware retriggerable. Only counters are hardware retriggerable. To simulate a retriggerable analog operation in software, set the task to respond to a trigger and then reconfigure the operation in software after it completes. Reconfiguring the task will allow it to respond to the next trigger. However, the time it takes to reconfigure the operation in software is completely system dependent and could cause the device to miss the next trigger if it does not reconfigure quickly enough.

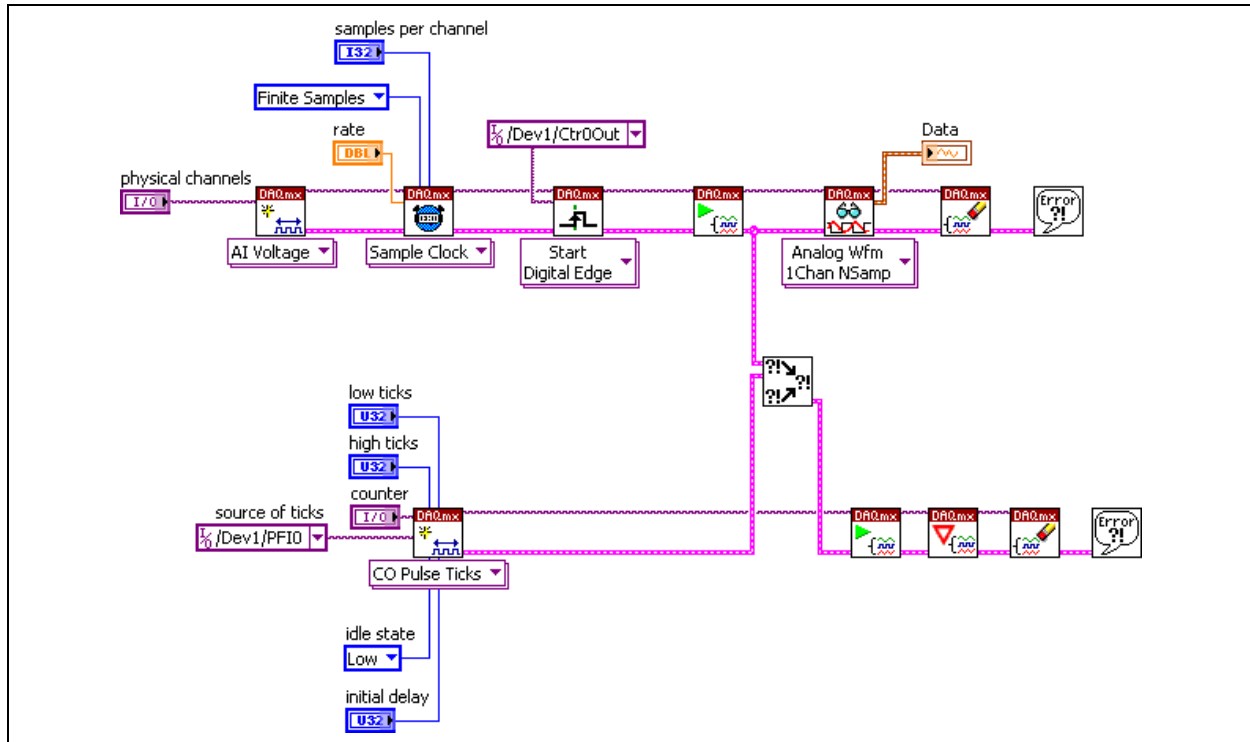
Since counter operations are retriggerable, they can be used to implement retriggerable analog input and output operations. Two counters are used to create a retriggerable finite pulse train to be used as the analog sample clock. The first counter is configured to create a continuous pulse train and the second counter is set up to perform retriggerable finite pulse generation (as we did in Exercise 9-5). This pulse is used to gate or pause the continuous pulse train generation of the first counter. Therefore, when the trigger occurs, the second counter creates a finite pulse which allows the continuous pulses to be generated on the first counter for as long as the finite pulse is still high.

The following example demonstrates how to perform a retriggerable analog input acquisition using counters. You do not have to explicitly set one counter for continuous pulse generation and another for a retriggerable finite pulse generation. NI-DAQmx will set this up for you when your VI configures one counter for finite pulse train generation and uses a property node to set it to retriggerable.



Event Triggering

There are cases in which you may want to skip a certain number of trigger pulses before starting an acquisition or generation. This type of trigger is referred to as an “event trigger” since you are triggering on the Nth ($2 < N < \text{Terminal Count}$) trigger event. Such an application can be implemented using the counters on a National Instruments E Series DAQ board. The following example demonstrates how to configure your DAQmx device for event triggering.



The top row configures the sample clock and the trigger. The bottom row configures the counter to divide down the signal so that it will output the “trigger” pulse when it inputs its “Nth” pulse. Notice the instance of the DAQmx Create Virtual Channel VI - **Counter Output»Pulse Generation»Ticks**. The high ticks and low ticks inputs are used to determine the “Nth” pulse. The source of the ticks can be a PFI line, an internal timebase, or any other appropriate signal source.

Using the Analog Input or Output Sample Clock to Gate a Counter

Another common application is to use the AI Sample Clock or AO Sample Clock to gate a buffered event counting or period measurement. To set up this configuration, use the DAQmx Export Signal VI to export the AI or AO Sample Clock to a PFI line. The counter will then use the PFI line as its gate.

C. Multiple Device Synchronization

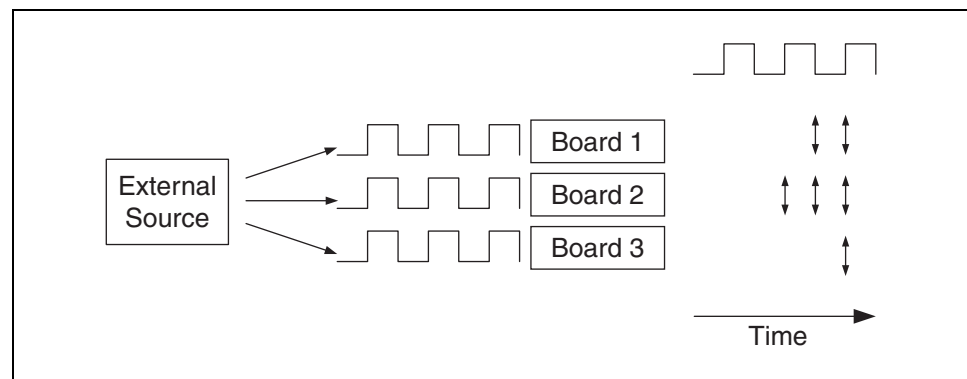
Board synchronization correlates measurements between multiple boards. Numerous applications are aided by a board's ability to synchronize itself. For example, by sharing timing signals, analog measurements can be taken in conjunction with counters by routing the sample clock to the counter to tell it when to latch a value.

Synchronization is especially important in high speed applications when a high channel count is needed. Either the boards will need to synchronize to an external device, and/or a master/slave relationship is established where one board is controlling the timing for all the boards in the measurement system.

There are several methods available for synchronization. The methods include using an external clock, using the internal timing bus, or using a Phase-Lock-Loop circuit. The latter method is beyond the scope of this course. However, the effect of delay and jitter become essential considerations when choosing a method for board synchronization.

External Signal Connections

One method of synchronization is using an external clock that allows the multiple boards to synchronize to this external source. Using this method, the board timing of the boards receiving the external source take on the accuracy and stability of the external clock source.



Using this method of board synchronization, the synchronization error will result from three sources. These sources include the length of the signal path, the individual board timing, and jitter. The combination of these factors will cause each board to see and respond to the external signal at different times.

For high frequencies, several factors make this method less than ideal. The clock signal will begin to deteriorate at frequencies around 5–10 MHz depending on the cabling used to wire between the external clock source and the boards. In addition, there is a transmission latency that will introduce pronounced phase delays at high speeds depending upon the length of the signal path. These delays will depend upon the capacitive, inductive, and resistive properties of the cabling. All timing sources will introduce jitter into the system that could become significant when attempting to synchronize the boards.

Another issue with this method occurs when using a start trigger to simultaneously begin acquisition on all of the boards in a measurement system. Usually, each board will trigger within one to two ticks of the board clock upon receiving the start trigger. This can cause phase delays between the boards.

To eliminate phase delays from transmission latencies and triggering, use the RTSI bus to transfer signals. The RTSI bus improves the limitations of using external wiring.

RTSI Bus

As discussed earlier, the RTSI bus is high-speed digital bus designed to facilitate systems integration by low-level, high-speed, real-time communication between National Instruments devices. Using RTSI, you can connect your DAQ device to motion, image acquisition, or digital I/O devices without consuming bandwidth on the host bus, such as the PCI bus. The RTSI bus also has built-in switching, so you can route signals to and from the bus on-the-fly through software.

Most NI DAQ devices support RTSI. In traditional plug-in DAQ devices, such as an E Series PCI device, the actual RTSI connection is accomplished with a special RTSI cable that is manually plugged into the RTSI pins on each device. The RTSI bus interface on a PCI DAQ device is an internal 34-pin connector where signals are shared via a ribbon cable inside the PC enclosure. RTSI cables are available for chaining two, three, four, or five devices together. RTSI functionality varies depending on device type, so always check your device documentation before beginning programming with RTSI.

The RTSI bus has eight lines available to users for sharing timing and triggering signals. Pins 0 to 6 are available for user signals, but pin 7, the RTSI Clock, is reserved for passing device clock signals between devices. Refer to ni.com/catalog for more information about the RTSI cable

If using a PXI module, the RTSI bus is built into the PXI chassis as the PXI Trigger Bus on the J2 backplane connector. The PXI Trigger Bus is standardized by the PXI Systems Alliance, now supported by a range of industry leaders. Thus, every PXI DAQ module you insert into a PXI chassis has the same built-in timing connections to every other PXI DAQ module in that chassis. No additional cabling is required. Refer to www.pxisa.org for more information about the PXI Systems Alliance.

Capable of passing timing signals of up to 20 MHz before signal deterioration, RTSI succeeds at high speed applications. However, phase delays due to triggering and transmission latencies will become more pronounced at high frequencies. To adjust for these issues, the use of a Phase-Locked-Loop (PLL) circuit allows for true multiple board synchronization. PLL is beyond the scope of this course.

Programming with RTSI

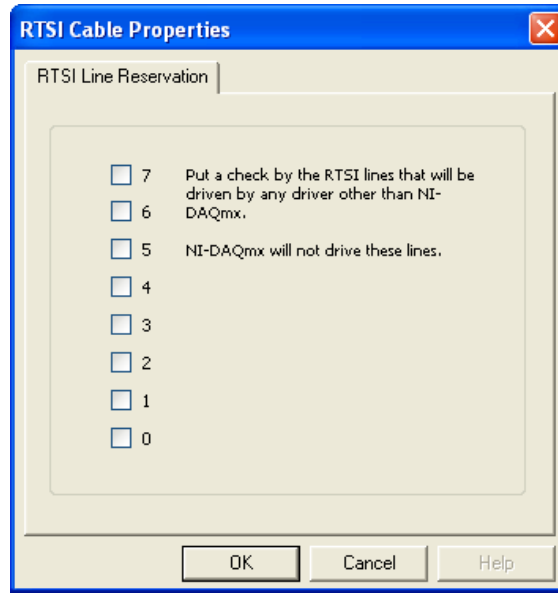
NI-DAQmx manages the majority of signal routing through the RTSI bus. However, you are still responsible for telling NI-DAQmx how the devices are connected to one another through internal buses. For PCI systems, you must register the RTSI cable in MAX, as was done in Exercise 10-2. For PXI systems, you must identify the type of PXI chassis in use.



Note To bypass NI-DAQmx management of routing signals, you can explicitly route signals to PFI or RTSI lines using the DAQmx Export Signal VI.

If two different applications try to drive the same RTSI line, damage to the board can occur. Since NI-DAQmx manages the routing of RTSI lines for you, it is not always known which RTSI are available for use. If another driver, such as the Traditional NI-DAQ driver in another application drives a RTSI line, NI-DAQmx will have no knowledge of this occurring. If both applications attempt to drive the same RTSI line, problems can and will occur.

To prevent the problem of double-driving a RTSI line, you can reserve certain RTSI lines in MAX to prevent NI-DAQmx from using that line when automatically routing signals. For PCI devices, right-click the RTSI cable that connects your devices and select Properties. The following RTSI Cable Properties window appears, as shown in the following figure.



Select which RTSI lines should not be used by NI-DAQmx. For PXI devices, highlight the chassis under the PXI system (the chassis must be identified first), and select the **Triggers** tab.

With an E Series device, the following signals can be routed over RTSI to be shared between multiple devices.

- AI Start or Reference Trigger
- AI Convert Clock
- AI Sample Clock
- AO Sample Clock
- AO Reference Trigger
- GPCTRO Source, Gate, or Output

Multi-Device Use Cases

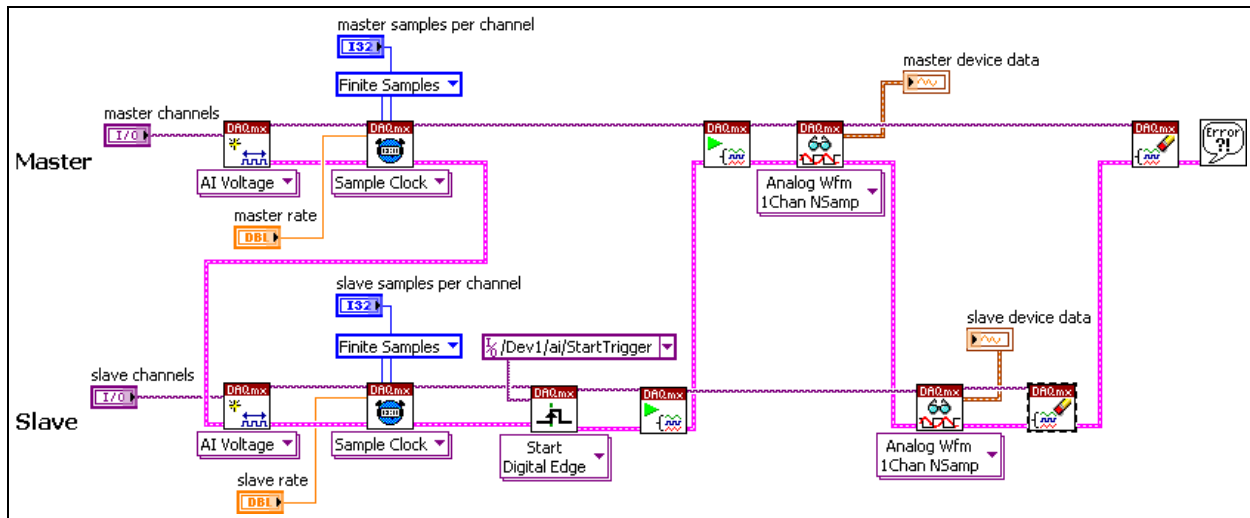
Much like single devices, the common use cases for simultaneous operations involving multiple devices are to simultaneously start operations on multiple devices by sharing a start trigger or to fully synchronize multiple device operations. A third use case is to start operations synchronized across multiple devices with a hardware start trigger. When synchronizing measurements, the analog input sample clock can be shared across all devices involved to synchronize all measurements to the same sample clock. Or, one of the board clocks of the device can be shared to synchronize the board clocks of all the devices involved.

Simultaneously Started Multiple Device Analog Input

To simultaneously start analog input operations across multiple boards, configure the master device to send its internal AI Start Trigger signals over the RTSI bus to start all of the slaves configured for a digital start trigger. The master can be started by a software call or configured for a hardware start trigger itself.

Unlike with single board operations, setting all of the sample clocks to be the same rate will not synchronize the operations. Each board is deriving its sample clock from its own oscillator clock, which is not guaranteed to be in phase with the oscillator clocks of the other devices.

The following example demonstrates how to simultaneously start multiple devices for analog input.



The slave device uses the AI Start Trigger of the master to start at the same time as the master device. Notice that the master device did not need to explicitly route its start trigger to RTSI. This route is made implicitly when the slave uses the DAQmx Triggering VI to digitally trigger off of this signal.

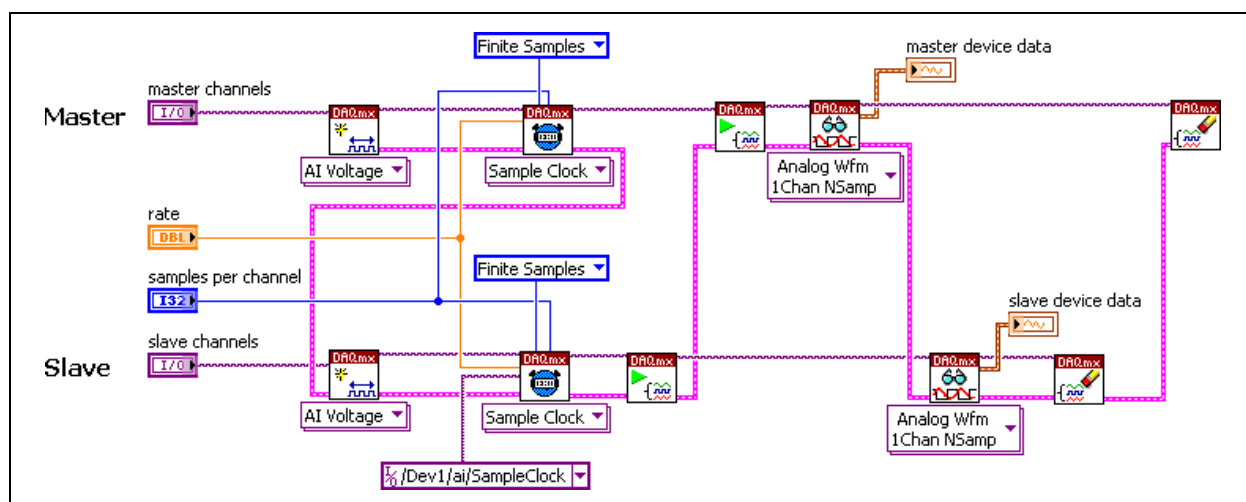
Synchronizing Multiple Device Analog Input Using a Shared Clock

One way of synchronizing multiple device analog input is to use the master sample clock for operations on all devices. The small propagation delays introduced between the master and slave due to routing the clock over RTSI are negligible with respect to the rates at which E Series devices operate.

If only the sample clock is shared between boards, each device is generating its own convert clock from its board clock. The board clocks that create the convert clock are free-running and are not synchronized to each other, and therefore, are not guaranteed to be in phase with one another. Thus, the

convert clocks will also be out of phase. This is more noticeable at higher sampling rates.

The example below demonstrates how to synchronize two analog input operations by sharing a sample clock in NI-DAQmx. Notice that the master device does not have to explicitly route its sample clock to a RTSI line. As long as a RTSI cable is configured or a PXI chassis is identified, NI-DAQmx will manage the routing for you. The slave device is configured to use the AI Sample Clock of the master using the DAQmx Timing VI. The slave must be started before the master, as seen by the propagating error lines.

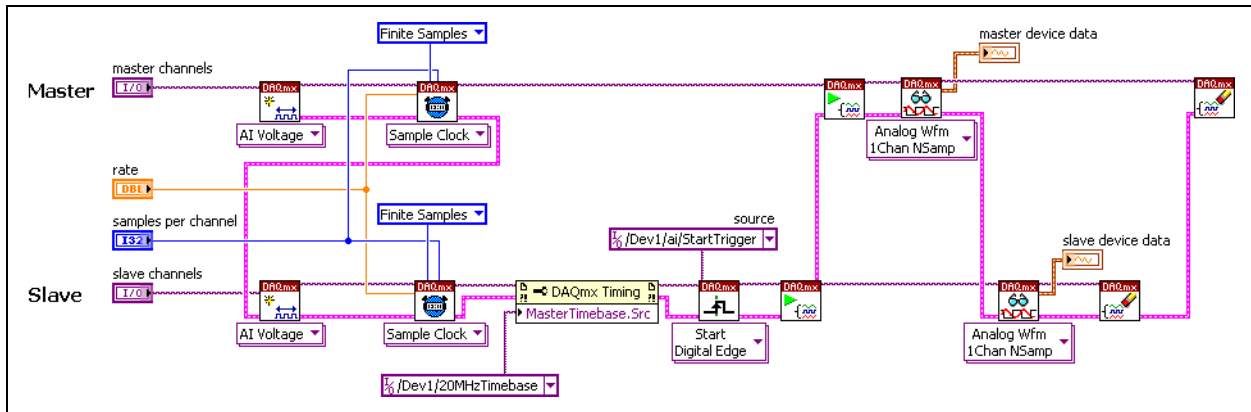


To fully synchronize multiple boards, you share both the sample clock and convert clocks. However, a better method would be to synchronize all devices to a common timebase, set each device to acquire at the same rate, and then start them simultaneously.

Synchronizing Multiple Device Analog Input Using a Shared Timebase

To achieve true multiple device synchronization, you must synchronize all of the devices to a common timebase in addition to providing a common trigger. Since E Series boards are not capable of phase-locking to a common reference clock, you must use the RTSI clock line to pass a common reference clock to all E Series devices. For multiple analog input operations, all slave devices would replace their board clocks with the master board clock sent over RTSI 7. Each device would set its sample clock to the same rate and the master would send a trigger to simultaneously start all devices. Because all devices are deriving their sample clocks from a common timebase clock and all acquisitions are started at the same time, the sample clocks will be fully synchronized to one another.

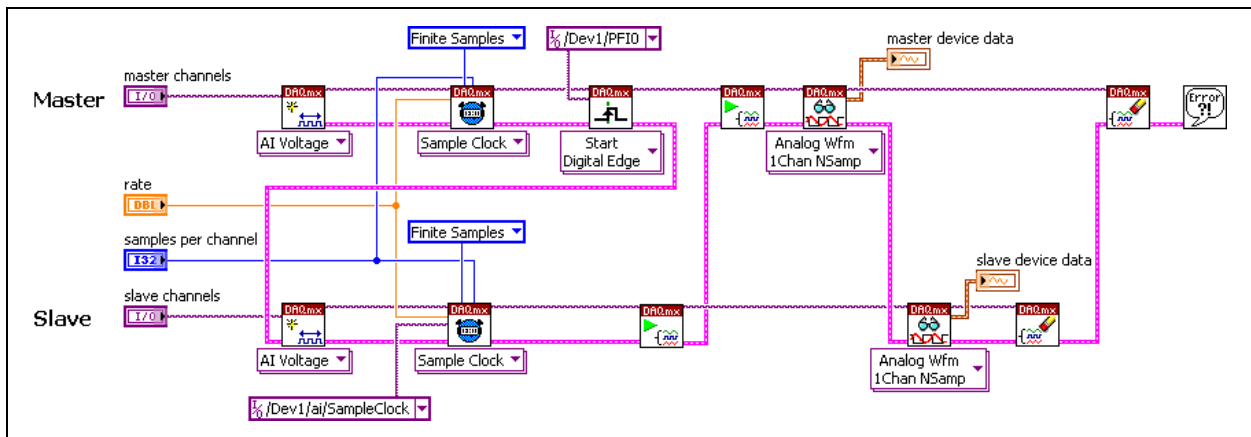
The following example demonstrates how to synchronize two devices performing analog input operations by sharing a common timebase and start trigger. To share a timebase, the slave device uses the DAQmx Timing property node to replace the source of its master timebase with the 20 MHz timebase of the master device. The DAQmx Trigger VI is then used to start the slave when the master generates its internal AI Start Trigger. In this example, the master is started with a software call.



Synchronization Using a Hardware Start Trigger

You can create a synchronized multiple device analog input operation with a hardware start trigger by configuring the master device in either of the two synchronization methods previously discussed for a hardware start trigger.

The following example demonstrates two analog input operations synchronized with a common sample clock and started with a hardware start trigger. The master is configured for a hardware start trigger using the DAQmx Trigger VI and the slave is configured to use the sample clock of the master.



Synchronization and Interoperability

Many applications involve simultaneous operations of devices in different families. These use cases are similar to the ones we have previously explored and include such things as synchronizing analog input or output of an MIO device with digital input or output, using a counter to create a clock for an analog or digital operation, synchronizing counter, and more. When not all devices involved are supported in NI-DAQmx, you have two options for development. You can develop the application entirely in Traditional NI-DAQ or use Traditional NI-DAQ to program the devices that are not supported in NI-DAQmx and use NI-DAQmx to program the rest. Because Traditional NI-DAQ and NI-DAQmx are two separate drivers and APIs, they can be used side-by-side in an application much like NI-Motion and NI-IMAQ can be used side-by-side. However, you cannot use both drivers to program the same device at the same time.

Using the drivers together is called interoperability mode.

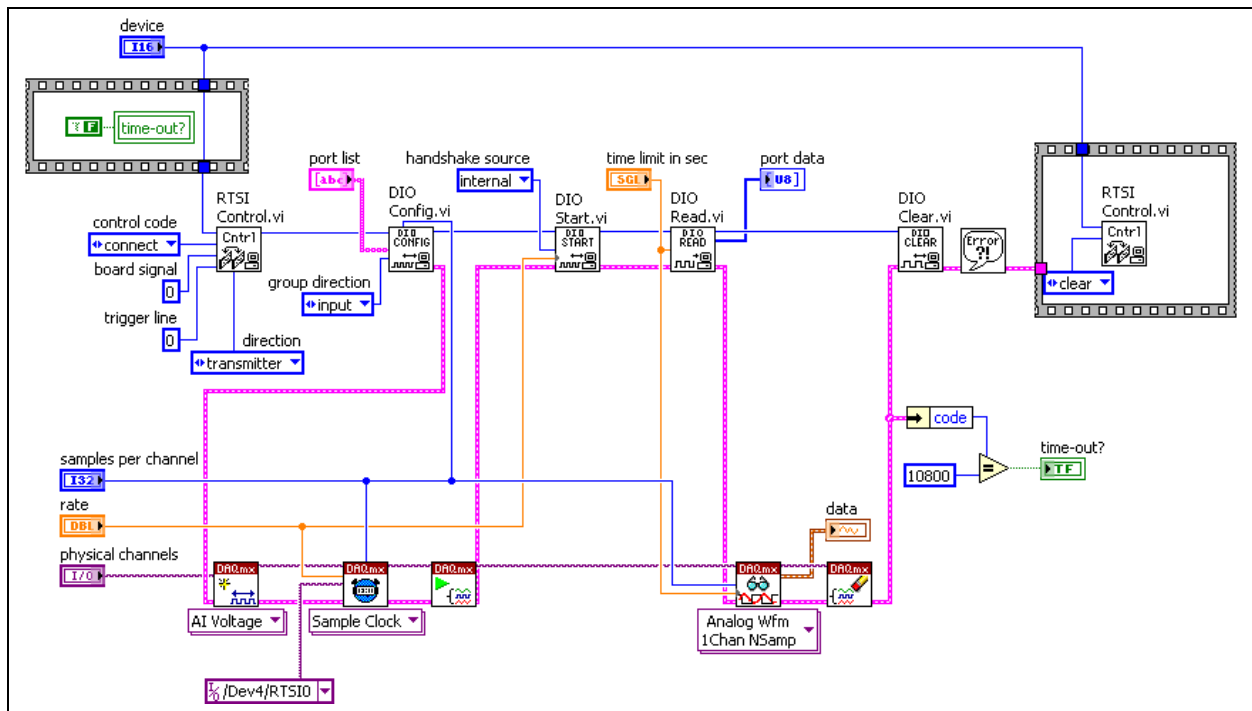
Using Traditional NI-DAQ and NI-DAQmx Together

Existing applications developed with earlier versions of NI-DAQ will run under Traditional NI-DAQ. However, earlier versions of NI-DAQ can *not* be used with NI-DAQmx on the same computer. You can use Traditional NI-DAQ and NI-DAQmx on the same computer with the following restrictions:

- Only one API can control a given device at a time.
- After using a device in the NI-DAQmx API, you must unreserve all NI-DAQmx tasks that are using that device before you can use that device through the Traditional NI-DAQ API. To unreserve NI-DAQmx tasks, call the DAQmx Stop Task VI or the DAQmx Clear Task VI.
- After using a device in the Traditional NI-DAQ API, you must reset the device before you can use that device in the NI-DAQmx API. To do this, call the Traditional NI-DAQ Device Reset VI. For SCXI devices, you must reset the communicator device by calling the Traditional NI-DAQ Device Reset VI.
- Another way to reset the device is to right-click the device in the Traditional NI-DAQ Devices folder in MAX and select **Reset**.
- To reset all devices in Traditional NI-DAQ and make them available for use in NI-DAQmx, right-click the Traditional NI-DAQ Devices folder in MAX and select **Reset Driver** for Traditional NI-DAQ.

- At any point in time, you must use all SCXI devices in the same chassis or in any of the chassis that are chained together through the same API. This restriction exists because SCXI modules in the same chassis or in the same set of chained chassis are all programmed by a single SCXI communicator device. That device can be operated only via one API at a time.

The following example demonstrates how to synchronize an analog input and digital pattern input operation using both Traditional NI-DAQ and NI-DAQmx. The NI 653x device is the master, sending its REQ clock to the MIO board over RTSI 0 using the RTSI Control VI. The MIO device is set up for an external clock on RTSI 0 using the DAQmx Timing VI. The MIO device is device 4. Therefore, to receive its clock from the RTSI bus, it need only connect to its own RTSI line.



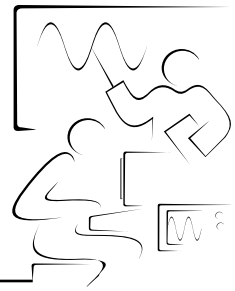
Summary

- You can simultaneously start operations using the AI Start Trigger VI.
- You can synchronize operations by routing internal signals.
- Use RTSI to transfer signals between multiple devices.

Notes

Notes

Appendix A



This appendix contains the following sections of useful information for DAQ system users:

- Theory of Common Transducers
- Analog I/O Circuitry

A. Theory of Common Transducers

Selecting your transducer is one of the first steps in building a DAQ system. The transducer you choose may affect the other components in your system. For example, some transducers require external signal conditioning for excitation or amplification, and others can make use of the LabVIEW VIs that convert voltage readings into units of temperature or strain. It is important to understand how different transducers operate and to know their advantages and limitations. Below is a discussion of the most common types of transducers: thermocouples, RTDs, thermistors, and strain gauges.

Thermocouples

One of the most frequently used temperature transducers is the thermocouple. Thermocouples are very rugged and inexpensive and can operate over a wide temperature range. Thermocouples can tolerate temperatures of several hundred degrees without degradation. Sensors, usually referred to as semiconductor sensors, can seldom function above 70 °C. Also, thermocouples are physically small and can rapidly track temperature changes.

A thermocouple is created whenever two dissimilar metals touch to produce a small open-circuit voltage as a function of temperature. This thermoelectric voltage is known as the Seebeck voltage (after Thomas Seebeck, who discovered it in 1821). The voltage is approximately linear for small changes in temperature, or

$$\Delta V \approx S\Delta T$$

where ΔV is the change in voltage, S is the Seebeck coefficient, and ΔT is the change in temperature. S varies with large changes in temperature, however, causing the thermocouple output voltages to be nonlinear over their operating ranges, as shown in Figure A-1. For this reason, you must use either polynomials or look-up tables to determine the voltage for any given temperature.

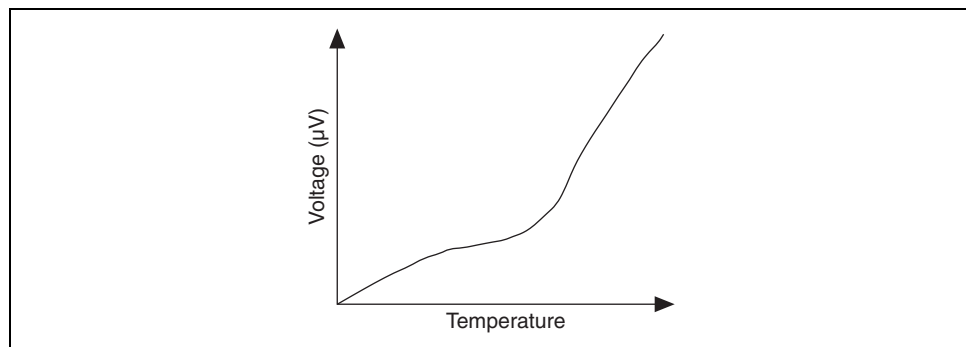


Figure A-1. Thermocouple Temperature versus Voltage Plot

Several types of thermocouples are available. These thermocouples are designated by capital letters that indicate their composition according to American National Standards Institute (ANSI) conventions. For example, a J-type thermocouple has one iron lead and one constantan (a copper-nickel alloy) lead.

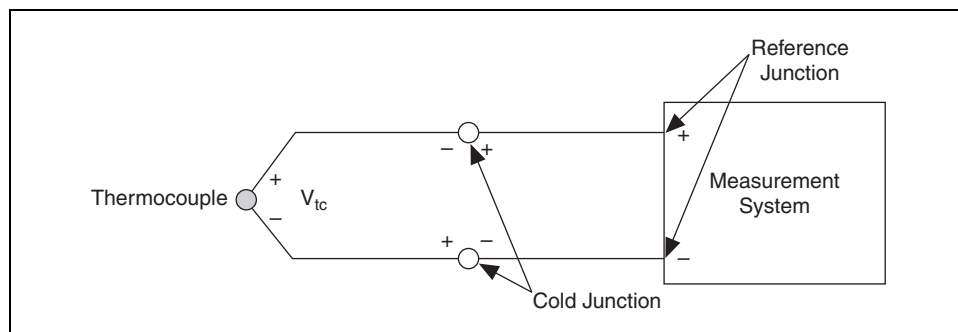


Figure A-2. Thermocouple System

Figure A-2 shows a thermocouple. The point at which the thermocouple connects to your measurement system is called the reference junction. When you connect the thermocouple leads to your measurement system leads, you create two additional junctions of dissimilar metals, called cold junctions. These cold junctions induce a thermoelectric voltage in your system. The process of eliminating this voltage is known as cold junction compensation (CJC). You can implement CJC in hardware or software. Both methods require that you measure the temperature at the reference junction with a sensor.

Table A-1 lists some common types of thermocouples. The table identifies the two dissimilar metals that form each thermocouple. The table also shows the valid temperature range for each thermocouple.

Table A-1. Common Types of Thermocouples

Thermocouple Type	Conductor		Temperature Range (°C)	Voltage Range (mV)	Seebeck Coefficient (μV)/(°C)
	Positive	Negative			
E	Chromel	Constantan	-270° to 1,000°	-9.835 to 76.358	58.70 at 0 °C
J	Iron	Constantan	-210° to 1,200°	-8.096 to 69.536	50.37 at 0 °C
K	Chromel	Alumel	-270° to 1,372°	-6.548 to 54.874	39.48 at 0 °C
T	Copper	Constantan	-270° to 400°	-6.258 to 20.869	38.74 at 0 °C
S	Platinum-10% Rhodium	Platinum	-50° to 1,768°	-0.236 to 18.698	10.19 at 600 °C
R	Platinum-13% Rhodium	Platinum	-50° to 1,768°	-0.226 to 21.108	11.35 at 600 °C

Thermocouple issues:

- Inexpensive and rugged
- Tolerate high temperatures
- Can track rapid temperature changes
- Requires CJC
- Very small voltages ($7\ \mu\text{V} - 40\ \mu\text{V}$ per degree C)—needs amplification
- Nonlinear output—measured voltages must be linearized

Resistive Temperature Devices (RTDs)

An RTD is a device whose resistance varies with temperature, as shown in Figure A-3. RTDs are available in different materials and resistance ranges, the most popular being the $100\ \Omega$ platinum RTD. RTDs are generally more accurate than thermocouples, and do not require cold-junction compensation. However, RTDs are typically more expensive, require linearization for accuracy, and may be affected by lead resistance (for example, long leads).

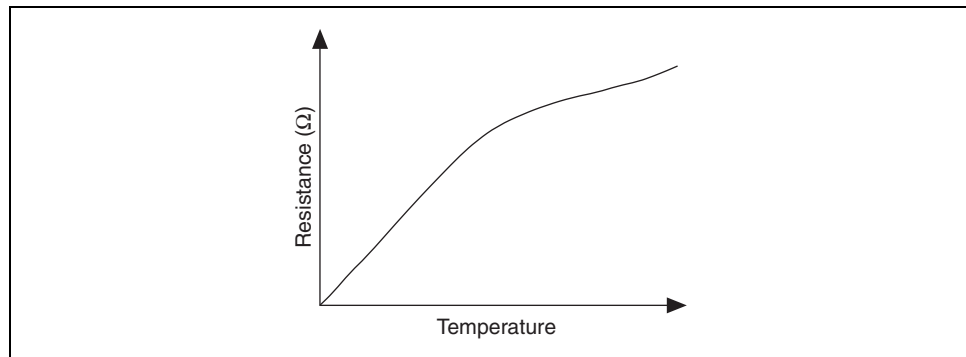


Figure A-3. RTD Temperature versus Resistance Plot

RTDs are available with two, three, or four leads. A two-wire RTD is the simplest, but may be inaccurate due to the wire lead resistance. The three-wire RTD uses a third wire to cancel out this lead resistance. The four-wire RTD (Figure A-4) is the most accurate because it also compensates for any error due to resistance matching of the leads. The RTD is similar to the strain gauge in that it requires excitation to induce a voltage across the RTD itself. The form of excitation is usually a current source.

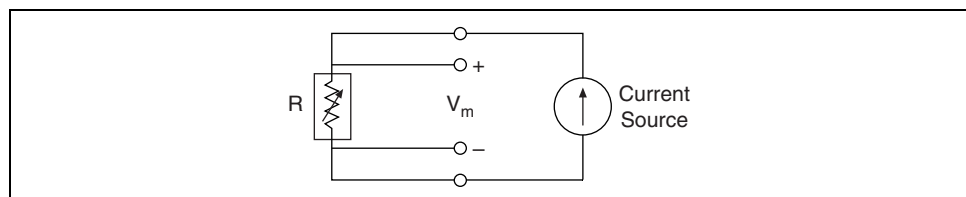


Figure A-4. Four-Wire RTD

In summary, RTDs have the following characteristics:

- More accurate, but more expensive than thermocouples
- Do not require CJC
- Require excitation—from signal conditioning hardware
- Require linearization
- Two-wire configuration is simple but has inaccuracies due to lead resistance
- Three-wire and four-wire RTDs use extra leads to minimize lead resistance

Integrated Circuit Sensors

An integrated circuit (IC) sensor is a temperature transducer made of silicon semiconductor material that acts as a temperature-sensitive resistor. IC sensors require an external power source. Although they are linear and cost efficient, they have slow response times and limited ranges. IC sensors are frequently used as CJC sensors for thermocouple measurements.

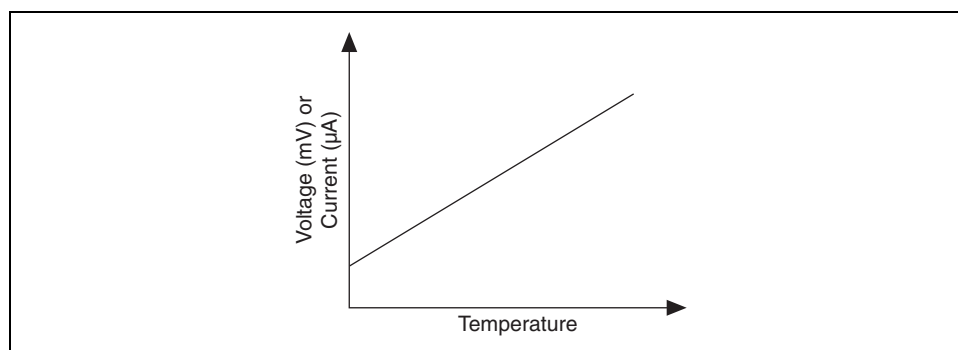


Figure A-5. IC Sensor Temperature versus Resistance Plot

Thermistors

A thermistor is a device whose resistance varies with temperature. As shown in Figure A-6, thermistors have a nonlinear output and require excitation. Because thermistors are relatively high-resistance devices, they do not require three-wire or four-wire configurations.

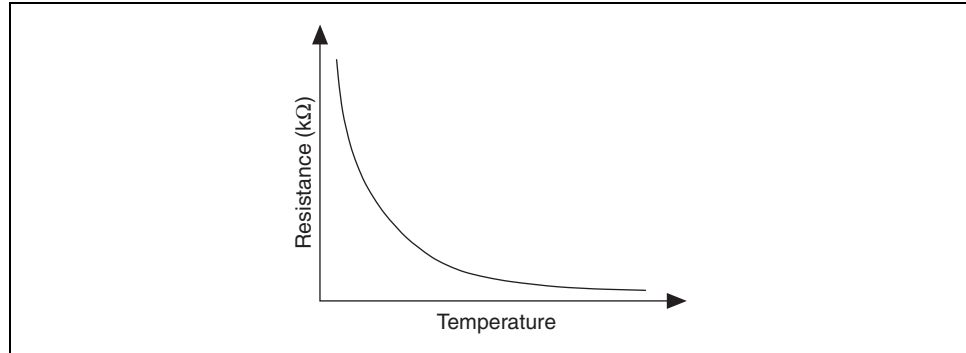


Figure A-6. Thermistor Temperature versus Resistance Plot

In summary, thermistors have the following characteristics:

- Require excitation—from signal conditioning hardware
- Require linearization
- High sensitivity
- High resistance—three-wire and four-wire configurations not required

Strain Gauges

A strain gauge is a device used to sense small movements in materials due to stress or vibrations. Strain gauges consist of thin conductors attached to the material to be stressed. Changes in the resistance of the gauge indicate deformation of the material. A variety of transducers use strain gauge elements mounted on diaphragms or other configurations to sense various physical quantities. For example, load cells are strain gauges configured to measure weight.

You usually use strain gauges in a configuration of resistors referred to as a Wheatstone bridge. In a bridge, four resistors are placed in a diamond configuration. When you apply a voltage to the bridge, the differential voltage (V_m) at the intermediate nodes varies as the resistor values in the bridge change. The strain gauge typically supplies the variable resistors.

Strain gauges come in different bridge configurations—full-bridge, half-bridge, and quarter-bridge. For a full-bridge strain gauge, the four resistors of the Wheatstone bridge are physically located in the strain gauge itself. For a half-bridge strain gauge, variable strain gauge elements usually occupy two arms of the Wheatstone bridge, while the other two arms are resistors that you or the signal conditioning hardware supply. Figure A-7 shows a half-bridge strain gauge. R_1 should equal R_2 , and R_g is the strain gauge resistance value at rest. While Figure A-7 shows a DC Voltage source providing excitation, some strain gauges require current excitation.

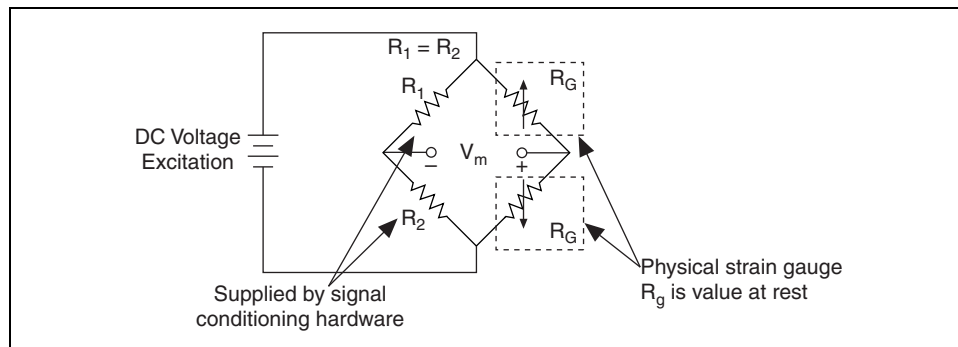


Figure A-7. Half-Bridge Strain Gauge

B. Analog I/O Circuitry

To help you better understand how a DAQ system converts real-world voltages into digital values that your computer can display and analyze, this section discusses the analog input components of your DAQ device.

Instrumentation Amplifier

To ensure maximum accuracy in the ADC, the instrumentation amplifier applies gain to the input voltage to coerce it to the range of the ADC. Jumpers and/or software configure the range for the ADC and LabVIEW determines the gain to apply according to the input signal limits. After amplifying a signal, the amplifier output fluctuates for a period of time, called the settling time, before stabilizing within an acceptable range of the actual voltage. The settling time affects how fast you can accurately sample data depending on the gain that was applied. Figure A-8 shows the settling time characteristics of a standard instrumentation amplifier. Notice the settling time, t_s , that passes before the amplified signal stabilizes within the acceptable voltage range ($\pm\% V$). The amplifier output should be allowed to settle before the ADC begins conversion. If the sampling rate (the rate at which the mux switches channels) exceeds the settling time of the instrumentation amplifier, the acquired data may be incorrect.

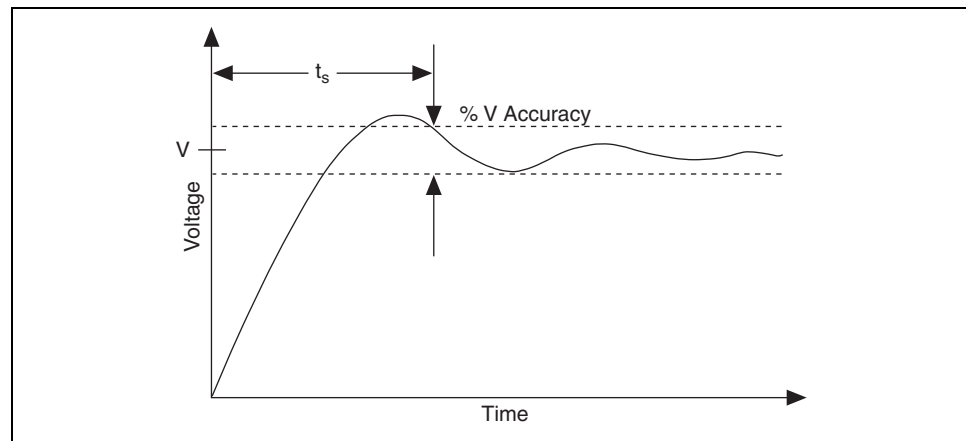


Figure A-8. Settling Time

For example, consider a DAQ device with two input signals as shown in Figure A-9. The signal connected to CH0 is at +5 V and the signal at CH1 is at -5 V. This is a worst-case situation where, when the mux switches, the instrumentation amplifier sees a “step” of 10 V. Assume the settling time for the instrumentation amplifier is 10 μ s and you are sampling at 200 ksamples/s. The amplifier is not given time to settle before the mux switches for the next reading. Hence, the voltages sent to the ADC for conversion are incorrect.

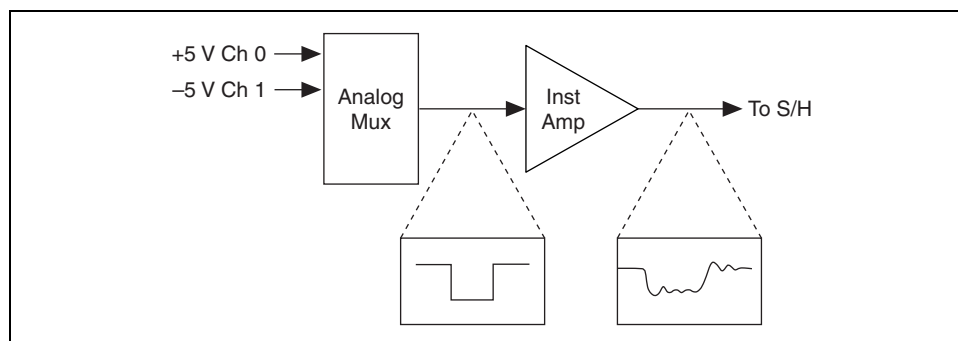


Figure A-9. Effect of Multiplexing Signals into an Instrumentation Amplifier

Figure A-10 shows the signals in Figure A-9 sampled both faster and slower than the settling time. Notice how the ADC digitized the wrong voltage level when the instrumentation amplifier did not have time to settle.

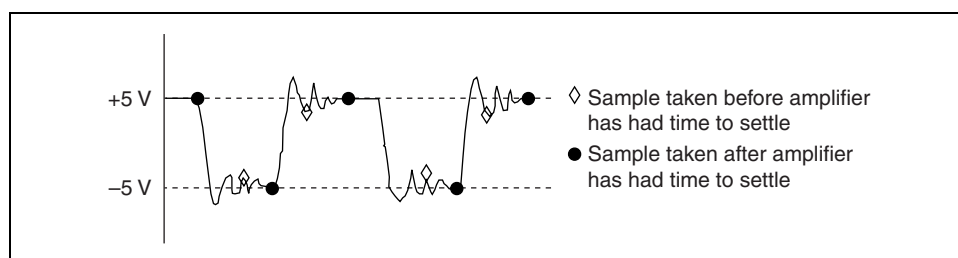


Figure A-10. Effect of Sampling Faster than the Instrumentation Amplifier Settling Time

This example is a worst-case situation—the instrumentation amplifier must swing full-scale. When consecutive input signals are within a certain percentage of each other, the amplifier settling time is much lower. For example, consider the following specifications for an AT-MIO-16X board. The instrumentation amplifier on this board settles to 16-bit accuracy in 40 μ s. You can accurately sample signals that switch between ± 10 V as fast as 25 kHz (worst-case situation). However, if the scanned input channels are within 10% of full-scale range of each other, the amplifier settles much faster and you can sample at the maximum board rate of 100 kHz.

Use the following tips to reduce the effects of settling time:

- Order the signal sampling so that the amplifier experiences the smallest voltage differences possible when switching from one channel to the next.
- Reduce the resistance and capacitance of wires to the DAQ board.
- Obtain a board that has an instrumentation amplifier with superior settling time. Many National Instruments DAQ boards use a custom instrumentation amplifier, the NI-PGIA, that settles equally at all gains. Figure A-11 compares the settling time of the NI-PGIA with an off-the-shelf instrumentation amplifier.

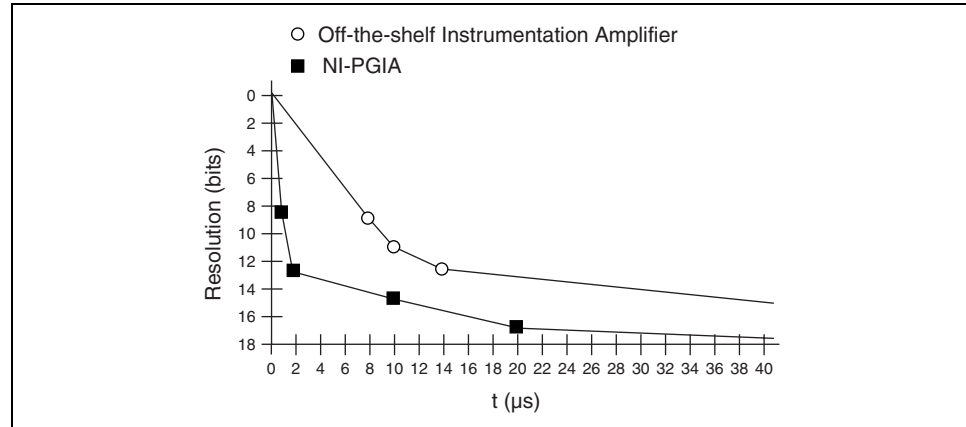


Figure A-11. Settling Time of NI-PGIA versus Off-the-Shelf Instrumentation Amplifier

DAQ device specifications are often in terms of least significant bit (LSB). One LSB is the voltage change increment corresponding to an LSB change in the digital value.

$$1 \text{ LSB} = \frac{\text{Input Range of ADC} / \text{Gain}}{2^{\text{number of ADC bits}}}$$

For example, the AT-MIO-16X can settle to ± 0.5 LSB (or $\pm 76.3 \mu\text{V}$) using a gain of 1 and an input range of 10 V.

$$\pm 0.5 \text{ LSB} = \left(\frac{10/1}{2^{16}} \right) * 0.5 = \frac{5}{65,536} = \pm 76.3 \mu\text{V}$$

ADC

The fundamental component of the analog input circuitry is the ADC. The ADC digitizes the analog input signal; that is, it converts the analog voltage into a digital value. The ADC stores the digital value in a FIFO buffer until it can be passed to computer memory. During high-speed acquisitions, the FIFO buffer prevents the loss of data due to interrupt latencies that may occur when transferring the data to computer memory.

DAQ devices may use different methods for performing A/D conversions. Some commonly used A/D techniques are successive approximation, flash, subranging (half-flash), integrating, and delta-sigma modulating.

Successive Approximation

The successive approximation ADC is the most popular type of ADC used on DAQ devices because it features high speed and high resolution at a modest cost. Figure A-12 shows an 8-bit successive approximation ADC.

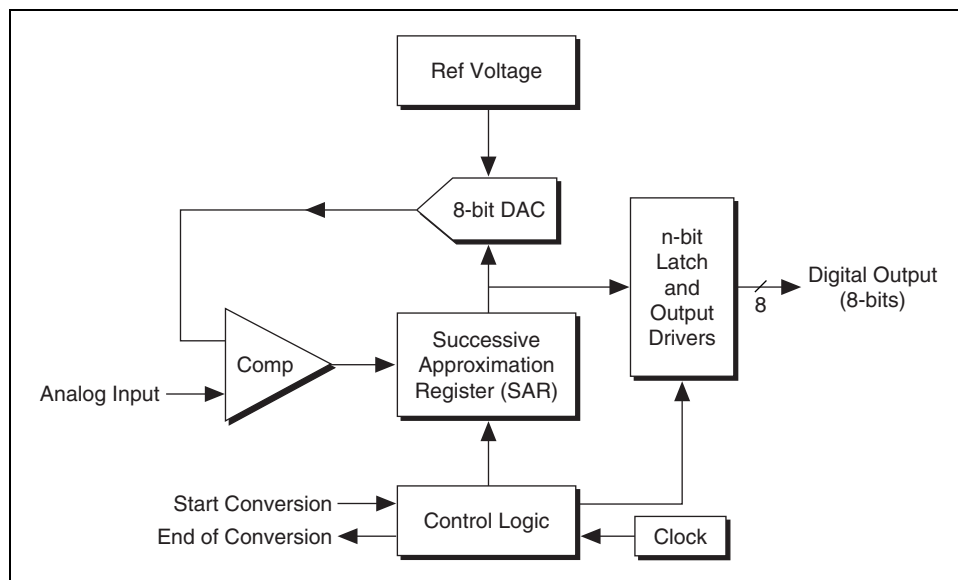


Figure A-12. Successive Approximation ADC

The successive approximation converter uses a technique similar to determining the weight of an object using standard weights. For example, consider that you have four weights—1 g, 2 g, 4 g, and 8 g. You place the unknown weight on one side of a scale and then place the largest known weight on the other side. If the scale does not tip, you add the next largest weight. If the scale tips, you remove the weight and try the next lighter weight. This is done until the scale balances. By totaling the weights put on the scale, you determine the weight of the object.

An 8-bit successive approximation converter works in a similar manner. The SAR initially sets all 8 bits of the DAC to 0. Then, starting with the most significant bit (MSB), each bit is set to 1 and the comparator (Comp) evaluates the output voltage. If the DAC voltage does not exceed the input voltage, the bit is left at 1; otherwise, it is set to 0. A digital code representing the input analog voltage is output after all n bits have been tested. For an 8-bit ADC, this process typically takes less than 2 μ s. Figure A-13 illustrates a conversion sequence for an 8-bit successive approximation ADC.

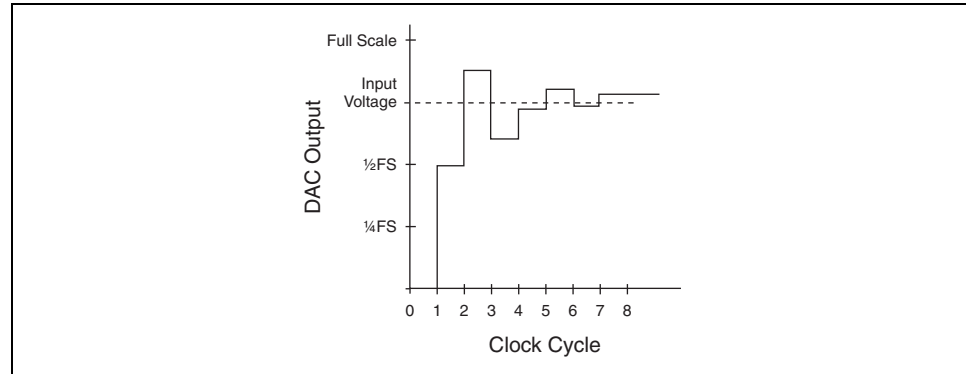


Figure A-13. Conversion Sequence of 8-Bit Successive Approximation ADC

Flash and Half-Flash

The fastest type of ADC is the flash ADC. For an n -bit ADC, the input voltage is applied simultaneously to 2^{n-1} comparators. As shown in Figure A-14, each comparator compares the input voltage to a different reference voltage. The reference voltages of consecutive comparators are one LSB apart. If the input voltage is greater than or equal to the reference voltage, the comparator outputs a 1; otherwise, it outputs a 0. The encoder translates the comparator outputs into a digital code.

Due to component cost and size, flash ADCs are typically available only with 8-bit or less resolution. Half-flash ADCs, a variation of the flash ADCs, use a hybrid technique that results in ADCs with lower cost and higher resolution, but slower conversion times than flash ADCs.

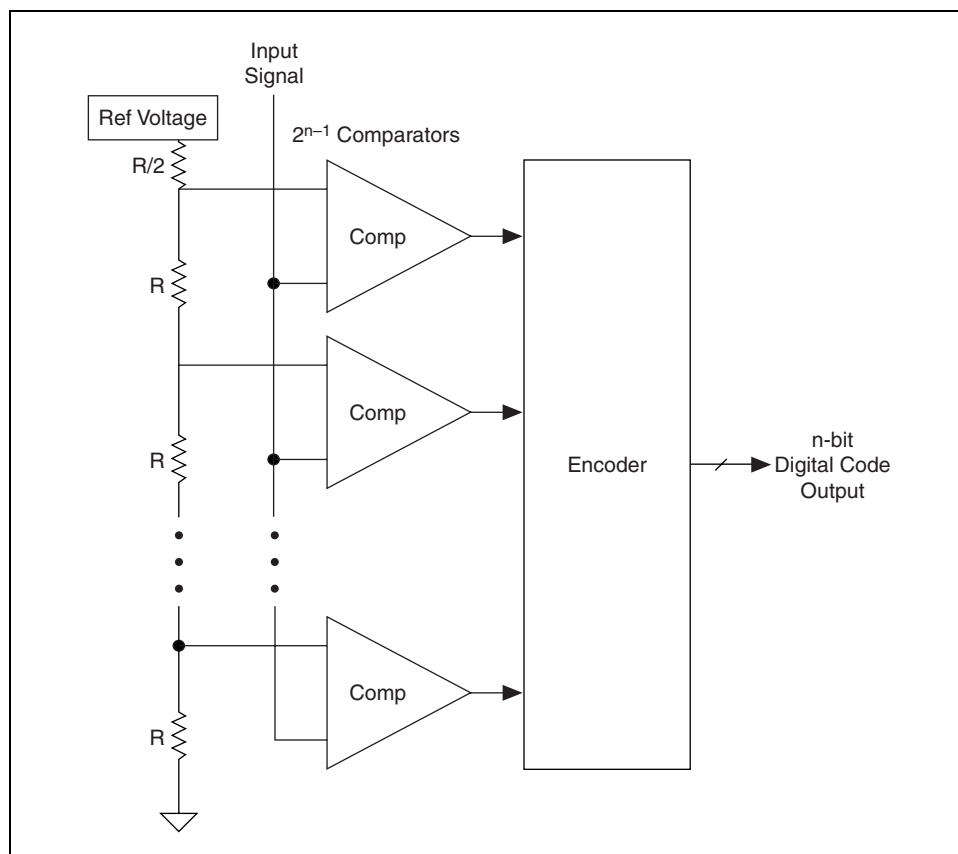


Figure A-14. Flash ADC

Integrating (Dual-Slope)

Another A/D conversion method, integrating, relies on integration to digitize the input signal. This type of ADC has several advantages—high resolution, good linearity, and input noise reduction using averaging. However, its main disadvantage is the slow conversion rate. Therefore, integrating ADCs are primarily used in digital multimeter and other slow measurement devices.

Delta-Sigma Modulating

The state-of-the-art technology in ADCs is the delta-sigma modulating ADC. These ADCs use delta-sigma modulators, combined with oversampling and digital filters to achieve high sampling rates, high resolution, and the best linearity of all ADCs. For example, this type of ADC delivers 16 bits of resolution at 48 kS/s with no differential nonlinearity.

Table A-2. Advantages/Use of Different Types of ADCs

Type of ADC	Advantages/Use
Successive Approximation	High resolution High speed Easily multiplexed Commonly used in DAQ boards DC signals
Flash	Highest speed Mature technology More expensive
Integrating	High resolution Good noise rejection Good linearity Mature technology Slow conversion rate Commonly used in DMMs
Delta Sigma	High resolution Excellent linearity Built-in anti-aliasing AC signals

Documentation Comment Form

National Instruments encourages you to comment on the documentation supplied with our products. This information helps us provide quality products to meet your needs.

Title: *Data Acquisition and Signal Conditioning Course Manual*

Edition Date: August 2003

Part Number: 320733K-01

Please comment on the completeness, clarity, and organization of the manual.

If you find errors in the manual, please record the page numbers and describe the errors.

Date manual was purchased (month/year): _____

Thank you for your help.

Name _____

Title _____

Company _____

Address _____

Email Address _____

Phone (____) _____ Fax (____) _____

Mail to: Customer Education
National Instruments Corporation
11500 North Mopac Expressway
Austin, Texas 78759-3504

Fax to: Customer Education
National Instruments Corporation
512 683 6837

Course Evaluation

Course _____

Location _____

Instructor _____ Date _____

Student Information (optional)

Name _____

Company _____ Phone _____

Instructor

Please evaluate the instructor by checking the appropriate circle. Unsatisfactory Poor Satisfactory Good Excellent

Instructor's ability to communicate course concepts

Instructor's knowledge of the subject matter

Instructor's presentation skills

Instructor's sensitivity to class needs

Instructor's preparation for the class

Course

Training facility quality

Training equipment quality

Was the hardware set up correctly? Yes No

The course length was Too long Just right Too short

The detail of topics covered in the course was Too much Just right Not enough

The course material was clear and easy to follow. Yes No Sometimes

Did the course cover material as advertised? Yes No

I had the skills or knowledge I needed to attend this course. Yes No If no, how could you have been better prepared for the course? _____

What were the strong points of the course? _____

What topics would you add to the course? _____

What part(s) of the course need to be condensed or removed? _____

What needs to be added to the course to make it better? _____

How did you benefit from taking this course? _____

Are there others at your company who have training needs? Please list. _____

Do you have other training needs that we could assist you with? _____

How did you hear about this course? NI Web site NI Sales Representative Mailing Co-worker

Other _____

